M.L. 16

Technical Report

INTER-LINGUAL SYNTAX BRACKETTING PROGRAMS

Cambridge Language Research Unit

Contract: AF 61(052)-542 M.L. 165

TECHNICAL REPORT

INTER-LINGUAL SYNTAX BRACKETTING PROGRAMS

by

D. S. LINNEY

"The research reported in this document has been sponsored by the Directorate Mathematical Sciences, AFOSR, through the European Office, Office of Aerospace Research, UNITED STATES AIR FORCE."

30th March 1963

Cambridge Language Research Unit
20 Millington Road
Cambridge England

INTER-LINGUAL SYNTAX BRACKETTING PROGRAMS

The work outlined in this report stems from the ideas of Dr. A.F. Parker-Rhodes, who constructed the first syntax dictionary for English which appears as Appendix I.

Sections I and III have been prepared jointly by D.S. Linney and R. McKinnon Wood, who is responsible for the final formulation of the bracketting Algorithm.

Section II is entirely due to R.McKinnon Wood.

Section IV was done by Y. Wilks, who also carried through the extension of the governor-dependence syntax dictionary with Miss E. Rigby, (Appendix II).

INTRODUCTION:

The object of a syntax-bracketting program is to enable a computer to analyse a sentence in the way that a child in high school does it. This activity, variously described as "syntax" in the U.S.A. and "analysis" in Great Britain, is hallowed by time and the grammarians. It is usually believed to elucidate the structure of a sentence, and uses as its fundamental notions subject, predicate, and the names of parts of speech, noun, verb, adjective, adverb, preposition, conjunction. Various notations may be used: C.F. Hockett (2) uses diagrams: but it is more convenient to use brackets. We shall use curved left-hand and right-hand brackets and permit their iteration.

This process results in a grouping or bracketting of all the words in a sentence which depends to a large extent on their syntactical and grammatical relations: a machine bracketting program must depend only on these relations; for a digital computer is not a human being.

The importance of such bracketting procedure for mechanical translation is considerable. Since the procedure depends on the machine having only syntactical information concerning the language, considerations of meaning are not

involved. An analysed or bracketted sentence may then be thrown into a standard form (e.g. with the governor at the end of the bracket-group) which will simplify the operation of a translation algorithm. Further, the bracket-groups will usually be semantical units on which semantical algorithms may be performed: and the translation of such units via a semantic interlingua is more likely.

Parker-Rhodes (9) writes:

> Even when two languages express themselves in an apparently quite different way, the standard form of their sentences is often identical. Thus consider
>
> The senior author was me ((the senior author)(me was))
> Moi j'etais l'auteur principal ((l'principal auteur) moi-j'etais))

The fundamental notions on which syntactical analysis depends are those of substituent and syntactic function, and the derivative notions of substituent type and participation class as expounded and defined in Parker-Rhodes (9) and (10) and in a series of publications by the C.L.R.U. : M.L. 136; M.L. 147; M.L. 154; M.L. 153. The traditional grammatical categories are thus replaced.

A substituent is any part of a sentence, a word or a phrase or a clause or the sentence itself, that is meaningful.

It is difficult in general, to assert that morphemes are substituents and only some punctuation marks can be so treated.

What is a correct machine analysis of a sentence?

These terms permit us to define recursively a valid bracketting of a sentence as an analysis such that

(1) A word is a substituent

(2) The set of substituents that form the group form a valid bracket under a computable algorithm: in other words, there is a function $\phi(X_1, X_2 \ldots X_n)$, where the values of the function are 0 and 1, and the bracket is valid, if and only if $\phi(X_1, X_2 \ldots X_n) = 1$

(3) The sentence is exhibited as a single substituent.

This definition permits the existence of syntactically ambiguous sentences, which reflects the actual situation in language. Another consequence is that a correct analysis of the sentence must exhaust all the substituents.

Apart from this definition, we make the restriction that valid brackets must consist of contiguous substituents. This may require re-ordering of substituents to be done prior to analysis, by unilingual routines. This condition derives from any computation which proceeds strategically from the inside of a sentence out, in other words which compares substituents and groups them: it would not hold for a

target strategy which attempted to break the sentence down from the outside in. Since the algorithm we have developed is of the first kind, target strategies are precluded.

Syntactical Properties:

The lattice framework of the Parker-Rhodes Model is used to provide a set of substituent types, which acts as a combinatorial frame into,which one may sort words according to certain properties. These properties are not themselves provided by the model, but by the language concerned. The properties considered in this study are

(1) Being a governor or being a dependent

(2) .Being initial and being final

(3) Concord or agreement with respect to grammatical number, i.e. being singular and being plural

(1) is discussed in M.L. 147 and is related to traditional grammatical notions of modification.

(2) is the fundamental property of the word-order of a bracket-group, and of a sentence. It is necessarily the case that speech, and thus (derivatively) written text, is simply ordered by the fact that it is a temporal process. It is conceivable that some written language might deliberately ignore order, but it is a fairly general fact that

languages do not - they make use of this order to eliminate syntactical ambiguity and to specify meaning. We shall make use of the idea of precedence and demonstrate its power.

Three obvious questions that suggest themselves are

(a) Can the substituent occur in the first place in substituent type i ?

(b) Can the substituent occur in the last place in substituent type i ?

(c) Can it occur elsewhere in substituent type i ?

It is clear, however, that the information derived from answers to these questions would not allow us to use all the relationships of precedence which may occur in a bracket group of more than three substituent elements. To exhaust the possibilities in a bracket group of n elements would require n questions. Since the property of preceding is defined to apply to finite and infinite sequences, it is better to have the properties of initiality and finality defined generally in terms of precedence. We define: an initial substituent is one that cannot be preceded by a substituent that is not initial. A final substituent is one that cannot precede one that is not final. First and last substituents may be defined as those that are preceded by none and that precede none of the substituents in a given bracket group. And since we are dealing with a simple

ordering, any two substituents will precede or be preceded one by the other.

The characteristic word-order of the sentence in a given language, which reflects the thought of the people who use it, will be given by the characteristic word-order possibilities which different substituent types have.

(3) Concord in general might be thought to be a property which consists in actual agreement, according to specifiable rule, of the shapes of signs, e.g. endings. It is true that this is the case with some languages, e.g. German, but even here it breaks down because of exceptions. In the case of English, the singularity or plurality of a word cannot be inferred from the grammatical shape at all: there is a difference between 'person' and 'persons' but not between 'sheep' and 'sheep', or 'class' and 'class'. Whether a word is marked as 'singular' or 'plural' depends on the person constructing the dictionary asking a question about the number of the word, and this does, in some sense clearly involve referring to a category 'number', which may be exemplified differently in different languages, and which could very easily be called a semantical category.

In the Parker-Rhodes syntax dictionary, M.L. 159, Appendix No. 1., no concord participation class has been

assigned to the substituent types, but this is not difficult: e.g. conjunct groups with "and" or "both....and" are always plural, and a nominal O-clause is always singular.

The essential point about any property that is to be of use in the analysis of a sentence is that the dictionary-maker should be able to assign a value 1 or 0, (or perhaps a probability value lying between 1 and 0) to the question, - does substituent i have the given property ? It is only of peripheral importance whether or not such a property, e.g. singularity - is "syntactical" or "semantical" under some arbitrary definition. It is clear that the properties we are using are close to traditional grammatical properties and so must be called syntactical: it is equally clear that the person making the dictionary entries is using semantical categories. This is not surprising when one realises that grammatical concepts, such as noun, verb, subject predicate, etc. have been defined traditionally with reference to semantical notions, such as naming, action, assertion.

If syntactical properties were defined, as Tarski (17) does, as properties that depend only on the kind, shape and order of the signs, then it could be argued that no properties, such as 'being a governor' or 'agreement with respect

to number or gender' are purely syntactical: a purely syntactical analysis would then depend solely on the property of word-order: this may be possible, but would, in the case of a number of languages, place restrictions on the strength of any syntactical bracketting algorithm which are insurmountable.

From the point of view of the bracketting algorithm which we have developed, any further property, to which the dictionary-maker can assign an answer without unreasonable uncertainty, may be added to the algorithm, or replace any section of it, if experience should show that it is more powerful. In this context then, distinctions between syntax and semantics become uninteresting. Incidentally, one may define a sentence purely syntactically - the set of substituents bounded by full-stops, - where 'full-stops' includes question marks and exclamation marks.

SECTION I.

The Bracketting Algorithm

The logical methods used in the presentation of the algorithm are derived from G. Boole (1) and (2) and are developed and expounded in Section 4 of this report.

1) We will assume the existence of a Syntax Dictionary for the language in question, consisting of a list of words attached to an appropriate Dictionary Entry. This entry will be divided into three sections giving the Governor-Dependent or G.D. entry, the Word-Order or W.O. entry and the Concord entry, respectively. This dictionary will also contain participation-class entries for the substituent types.

Each of these three entries is in turn subdivided into not more than 23 units, and in practice about 13 (chosen for the language in question); these units correspond to each of the 23 substituent types possible in the Parker-Rhodes lattice model. Each of these units contains, coded in a suitable number of bits, the information concerning the possibility of the participation of the word in that substituent type.

(a) The Governor-Dependent Entry:

Each unit in this section contains two bits, these being

the answers to the following two questions: (1) Can the word occur in that particular substituent type as a Governor? (2) Can the word occur in that particular substituent type as a Dependent?

The answer is coded as a '1' for 'Yes', and an '0' for 'No'. It is clear that a word with a G.D. unit reading 00 can occur as neither Governor nor Dependent; that is, it cannot occur at all in that substituent type. Further, a word with a G.D. reading 01 can occur only as a dependent, with a reading 10 can occur only as a governor, and 11 can occur as either.

For convenience in reading, these two bits are generally expressed as a quartal number, i.e. 00 is written as 0, 01 as 1, 10 as 2, and 11 as 3. For the purposes of calculation, and for purposes of logic here, the two bits are treated as independent binary variables.

(b) The Word-Order Entry:

Each unit in this section again contains two bits, giving the answers to the following questions, with the same binary coding: (1) Can the word occur in that particular substituent type as a 'Final' word? (2) Can the word occur in that particular substituent type as an 'Initial' word?

Here a 'final' word is defined to be one that may end a bracket group and may not precede an 'initial' word, and an 'initial' word is defined to be one that may start a bracket group and may not be preceded by a 'final' word.

As with the G.D. entry, we have four possibilities for the entry in each W.O. unit. Since we wish to give effect to the order of precedence which is the fundamental feature of word-order in a language, the four possible entries have the following significance:

00: The substituent cannot occur at all in that substituent type;

01: The substituent may precede any other substituent, but may not follow a substituent with the entry 10;

10: The substituent may be preceded by any other substituent but may not precede a substituent with the entry 01;

11: The substituent may precede or be preceded by any other substituent.

(c) The Concord Entry:

The number of bits required in this entry is dependent on the amount of concord information present in the language concerned: some languages may require distinctions of gender,

mood, person, number, etc. to be reflected. In the case of English, which we consider here, two bits are required; these are the answers to the questions:

(1) Can the number (in the normal grammatical sense) be plural in that substituent type, or can it be singular in the first or second person in that substituent type?

(2) Can the number be third person singular in that substituent type?

Here again the case 00 excludes the participation of the word and 11 is ambiguous (as is the case with 'sheep').

It is worth noting that a word may be plural in one substituent type and singular in another. For example, the word 'book' would be plural when used as a verb, but singular when used as a noun.

A Valid Bracket Group may be defined as follows:

A valid bracket group consists of two or more substituents, which are either words of the language or compound substituents representing other valid bracket groups, which satisfy three separate rules, viz., the G.D. Rule, the W.O. Rule and the Concord Rule.

Let a bracket group consist of n substituents, viz., S_1, S_2, ... S_i ... S_n where $n \geq 2$. The three rules may

then be formulated

(a) The G.D. Rule:

The bracket group is valid under this rule if and only if: for at least one substituent type, corresponding to one of the units of the G.D. section of the dictionary entry of the substituents; (1) for every substituent, the substituent must occur as a governor, or as a dependent, or both.

(2) For at least one substituent, the substituent must occur as a governor.

(3) For not more than one substituent, the substituent may occur as a governor but not as a dependent.

We treat the answers to the questions used in setting up the dictionary as propositions, which are true or false, and the dictionary coding gives us 1 or 0. We then apply Boolean propositional logic to the above rule.

Let the entry for any given one of the units in the G.D. section of the dictionary for substituent type S_i be $g_i \quad d_i$.

Then Rule (1) may be formalised as:

For all i, $(g_i \cup d_i)=1$.

Rule (2) may be formalised as:

For at least one j, $g_j = 1$

and Rule (3) as

There is not more than one value of k for which

$$(g_k \cap \bar{d}_k) = 1 .$$

We combine these rules and simplify, thus obtaining the complete G.D. Rule: the bracket group consisting of substituents S_1 , S_2 , ... S_i ... S_n is valid if and only if

$$d_1 \cap d_2 \cap \ldots \cap d_{i-1} \cap g_i \cap d_{i+1} \cap \ldots \cap d_m = 1, \ (i=1 \text{ to } n).$$

If we write out this formula in full for $n=3$ we have

$$(d_1 \cap d_2 \cap g_3) \cup (d_1 \cap g_2 \cap d_3) \cup (g_1 \cap d_2 \cap d_3) = 1$$

(b) The W.O. Rule:

A point of interest is that whereas the G.D. Rule is commutative, (i.e. no regard is paid to the order of substituents in a bracket group), this cannot be true of the W.O. Rule, because this is intended to reflect information about the order of words in a text sentence - the order of the substituents is now a factor. We adopt the convention

that in the representation of the bracket group as S_1 , S_2 , ... S_i , ... S_n , S_1 represents the last substituent, S_2 the second last, ... and S_n the first.

The bracket group is valid under this rule if and only if: for at least one substituent type

(1) The last substituent must be marked as final.

(2) The first substituent must be marked as initial.

(3) A substituent which cannot be initial must not be followed by a substituent which cannot be final.

(4) A substituent which cannot be final must not be preceded by a substituent which cannot be initial.

We again apply Boolean algebra, and representing the dictionary units in the W.O. section for substituent S as F_i I_i we may then formalize rule (1) as $F_1 = 1$

rule (2) as $I_n = 1$

rule (3) as for any j ,

where $\bar{I}_j = 1$, $F_{j-1} = 1$.

and rule (4) as for any k ,

where $\bar{F}_k = 1$, $I_{k+1} = 1$.

Combining these rules and simplifying, we obtain the complete W.O. Rule: A [redacted] bracket group consisting of substituents S_1, S_2, ... S_i ... S_n is valid if and only if

$$F_1 \cap \cdot \cap F_i \cap I_{i+1} \cap I_n = 1 \quad (i=1 \text{ to } (n-1))$$

If we write this out in full for the case n=3 we have

$$(F_1 \cap F_2 \cap I_3) \cup (F_1 \cap I_2 \cap I_3) = 1$$

(c) The Concord Rule:

The bracket group is valid under this rule if and only if for any substituent type, either (1) every substituent is marked as plural; or (2) every substituent is marked as singular; or both. (We have determined that first and second persons singular are marked as plural. Thus in 'I say', both 'I' and 'say' are treated as plural).

If we represent any given dictionary in the Concord section as $P_i \; S_i$,

then we may write the rule as

$$(P_1 \cap P_2 \cap \cap \cap P_i \cap \ldots P_n) \cup (S_1 \cap S_2 \cap \cap S_i \cap S_n) = 1$$

2) The Practical Algorithm

It is evident that the computation of these rules, with the exception of the Concord Rule, presents practical problems when considered for operation over the complete string

of substituents S_1, S_2, ...S_i ... S_n . Although the expression of these rules appears simple, it is necessary first to find the right value of i before the calculation can proceed, and it must also be borne in mind that the same calculation must be carried out for each of the substituent types present in the given languages before it is known whether a bracket group of a given length is or is not valid. As this form of the calculation is identical for each of the substituent types, using as variables the different units of the dictionary, there are clear advantages in allowing the calculation to proceed in parallel, where the computational possibilities of parallel calculation in modern computers can be exploited fully. Thus we may calculate in parallel up to 40 bits on the EDSAC II. or up to 960 bits on 80 column punched card equipment, thus obtaining a capacity for handling up to 20 substituent types on EDSAC II.

Logical operations on this type of equipment are carried out on two variables at a time; to compute $a \cup b \cup c$ it is necessary to compute $a \cup b = x$ and then compute $x \cup c$. We must therefore find a recursive algorithm, handling only two variables at a time, but operating in parallel over all the substituent types, for each of the three rules given above. In these algorithms, we must note that we do not know the value of i, where there is a discontinuity in the expression

of the rule, until we get there.

(a) The G.D. Algorithm:

Although this rule is commutative, the W.O. rule is not - thus a combined rule will not be. We will therefore adopt the same convention as that adopted for the first formulation of the W.O. rule, and consider an ordered bracket group S_1 , S_2 , S_i , S_n , where S_1 is the last substituent and S_n is the first. We again use the representation of any given dictionary unit as g_i d_i .

We first consider S_1 and S_2 and write a truth table for the 16 possibilities given by all possible combinations of g_1 , g_2 , d_1 , and d_2 , which will give the required result

g_1	d_1	g_2	d_2	g_o	d_o
1	1	1	1	1	1
0	1	1	1	1	1
1	0	1	1	1	0
0	0	1	1	0	0
1	1	0	1	1	1
0	1	0	1	0	1
1	0	0	1	1	0
0	0	0	1	0	0
1	1	1	0	1	0
0	1	1	0	1	0
1	0	1	0	0	0
0	0	1	0	0	0
1	1	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

On the left-hand side of this table we list the sixteen possible combinations of g_1, g_2, d_1, d_2, and on the right-hand side, the desired values of the two functions which we call g_o and d_o .

The function g_0 is the function required by the G.D. rule for a bracket group of only two members. The function d_0 is one which is =1 only when (1) $(g_1 \cup \bar{d}_1) \cap (g_2 \cup \bar{d}_2) = 1$

and (2) $(g_1 \cap \bar{d}_1) \cap (g_2 \cap \bar{d}_2) = 1$

i.e. referring to the description of the G.D. Rule, $g_0 = 1$ if the three rules (1) (2) and (3) are satisfied, and $d_0 = 1$ if Rule (1) is satisfied and the conditions for rule (3) have not arisen. We then have four possible outcomes of the calculation involving g_1, d_1, g_2, and d_2, viz: $g_0 d_0 =$ 00, 01, 10, 11. The case 00 arises either if Rule (1) is broken, i.e. S_1 or S_2 or both cannot participate or if Rule (3) is broken, i.e. both S_1 and S_2 have entries such that $g_i \cap \bar{d}_i = 1$ so that both S_1 and S_2 can only participate as governors in that bracket group. The case 01 arises when both S_1 and S_2 are dependents only and indicates that the bracket group is not valid, but might become valid if a further substituent S_3 were added which could act as governor. The case 10 indicates that the bracket group is valid, but will become invalid if a further substituent is added which can only act as a governor. And the case 11 indicates that the bracket group is valid, and will remain valid when a further substituent is added, either as a governor or as a dependent. A deliberate parallel has been

made here between the form of the function $g_0\ d_0$ and the dictionary unit $g_i\ d_i$, so that we may treat the result of our computation on S_1 and S_2 as a dummy substituent S_1 with dictionary reference $g_0\ d_0$, where g_0 and d_0 have exactly the same effect as a real dictionary entry g_i and d_i.

We may thus reduce the form of the G.D. algorithm from

$\chi\ (g_1\ d_1\ g_2\ d_2 \ldots g_i\ d_i \ldots g_n\ d_n) = 1$ to

$$\phi\left[\phi\left\{\left[\phi(g_1\ d_1\ g_2\ d_2)\right],\ g_3\ d_3\right\},\ g_4\ d_4\right] =$$

$g_0\ d_0$ where $\chi = g_0$, and this gives the required recursive algorithm.

We now require to reduce the truth table given above to two calculable expressions, one for g_0 and one for d_0. We first express the truth table in Hilbert's Exceptional Normal Form:

$$g_0 = (g_1 \cap d_1 \cap g_2 \cap d_2) \cup (\bar{g}_1 \cap d_1 \cap g_2 \cap d_2) \cup$$

$$\cup (g_1 \cap \bar{d}_1 \cap g_2 \cap d_2) \cup (g_1 \cap d_1 \cap \bar{g}_2 \cap d_2) \cup (g_1 \cap \bar{d}_1 \cap \bar{g}_2 \cap d_2) \cup$$

$$\cup (g_1 \cap d_1 \cap g_2 \cap \bar{d}_2) \cup (\bar{g}_1 \cap d_1 \cap g_2 \cap \bar{d}_2).$$

Simplifying by repeated application of the formula

$$\phi\ X \cup \phi\ \bar{X} = \phi$$

we obtain $g_o = (d_1 \cap g_2 \cap d_2) \cup (g_1 \cap g_2 \cap d_2) \cup (g_1 \cap d_1 \cap d_2) \cup$

$\cup (g_1 \cap d_1 \cap g_2) \cup (g_1 \cap \bar{d}_1 \cap d_2) \cup (d_1 \cap g_2 \cap \bar{d}_2) \cup$

$\cup (g_1 \cap \bar{g}_2 \cap d_2) \cup (\bar{g}_1 \cap d_1 \cap g_2)$

$= (d_1 \cap g_2) \cup (g_1 \cap d_2) \cup (g_1 \cap d_2) \cup (d_1 \cap g_2)$

$= (d_1 \cap g_2) \cup (g_1 \cap d_2)$

Similarly

$d_o = (g_1 \cap d_1 \cap g_2 \cap d_2) \cup (\bar{g}_1 \cap d_1 \cap g_2 \cap d_2) \cup (g_1 \cap d_1 \cap \bar{g}_2 \cap d_2) \cup$

$\cup (\bar{g}_1 \; d_1 \; \bar{g}_2 \; d_2)$

$= (d_1 \cap g_2 \cap d_2) \cup (g_1 \cap d_1 \cap d_2) \cup (\bar{g}_1 \cap d_1 \cap d_2) \cup (d_1 \cap \bar{g}_2 \cap d_2)$

$= (d_1 \cap d_2) \cup (d_1 \cap d_2)$

$= d_1 \cap d_2$

We may then state the G.D. algorithm in its final form: Given a string of substituents S_1 , S_2 , ... S_n , with dictionary entries $g_1 \; d_1$, $g_2 \; d_2$ ... $g_n \; d_n$, for any given substituent type,

Compute $g_o = (d_1 \cap g_2) \cup (g_1 \cap d_2)$ and $d_o = d_1 \cap d_2$

If $g_0=1$ $(S_1\ S_2)$ is a valid bracket group under the G.D. rule.

If $d_0=1$ and $g_0=0$ then $(S_1\ S_2)$ is not a valid bracket group under the G.D. Rule, but $(S_1\ S_2\ S_3)$ might be valid.

If g_0 and d_0 are both $=0$, S_1 and S_2 cannot belong together in a bracket group of that substituent type.

Unless g_0 and d_0 are both $=0$,

Compute $g_0' = (d_0 \cap g_3) \cup (g_0 \cap d_3)$

$d_0' = d_0 \cap d_3$

If $g_0'=1$ $(S_1\ S_2\ S_3)$ is a valid bracket group.

If $d_0'=1$ and $g_0'=0$, $(S_1\ S_2\ S_3)$ is not yet a valid bracket group, but might become so when S_4 is added.

If $g_0'=0$ and $d_0'=0$, S_1, S_2, S_3, cannot all three belong together in a valid bracket group of that substituent type.

Unless g_0' and d_0' are both $=0$,

Compute $g_0'' = (d_0' \cap g_4) \cup (g_0' \cap d_4)$, $d_0'' = d_0' \cap d_4$

Continue until some set n of substituents $S_1 \ldots\ldots S_n$ forms a valid bracket in some substituent type.

(b) The Word-Order Algorithm:

We now apply the same technique to the W.O. rule given before. We must first note, however, that we have the two rules given for the W.O. rule, viz. that $F_1 = 1$ and $I_n = 1$. We cannot fit both of these conditions felicitously into a recursive algorithm, in particular, if we start with substituent S_1 as we did for the G.D. algorithm, then we cannot fit this condition $F_1 = 1$ into a recursive algorithm, for it only applies to S_1. We will thus ignore this rule for the moment and add it to the algorithm later.

We construct a truth table as before, which satisfies rules (2), (3) and (4) of the Word-Order algorithm, and is recursive.

F	I	F	I	F	I
1	1	1	1	1	1
0	1	1	1	0	1
1	0	1	1	1	1
0	0	1	1	0	0
1	1	0	1	0	1
0	1	0	1	0	1
1	0	0	1	0	1
0	0	0	1	0	0
1	1	1	0	1	0
0	1	1	0	0	0
1	0	1	0	1	0
0	0	1	0	0	0
1	1	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

We may express this as

$$F_0 = (F_1 \cap I_1 \cap F_2 \cap I_2) \cup (F_1 \cap \bar{I}_1 \cap F_2 \cap I_2) \cup (F_1 \cap I_1 \cap F_2 \cap \bar{I}_2) \cup (F_1 \cap I_1 \cap F_2 \cap I_2)$$

$$= (F_1 \cap F_2 \cap I_2) \cup (F_1 \cap I_1 \cap F_2) \cup (F_1 \cap \bar{I}_1 \cap F_2) \cup (F_1 \cap F_2 \cap \bar{I}_2)$$

$$= F_1 \cap F_2$$

and

$$I_0 = (F_1 \cap I_1 \cap F_2 \cap I_2) \cup (\bar{F}_1 \cap I_1 \cap F_2 \cap I_2) \cup (F_1 \cap \bar{I}_1 \cap F_2 \cap I_2) \cup (F_1 \cap I_1 \cap \bar{F}_2 \cap I_2)$$

$$\cup (\bar{F}_1 \cap I_1 \cap \bar{F}_2 \cap I_2) \cup (F_1 \cap \bar{I}_1 \cap \bar{F}_2 \cap I_2)$$

$$= (I_1 \cap F_2 \cap I_2) \cup (F_1 \cap F_2 \cap I_2) \cup (F_1 \cap I_1 \cap I_2) \cup (\bar{F}_1 \cap I_1 \cap I_2)$$

$$\cup (F_1 \cap \bar{I}_1 \cap I_2) \cup (I_1 \cap \bar{F}_2 \cap I_2) \cup (F_1 \cap \bar{F}_2 \cap I_2)$$

$$= (I_1 \cap I_2) \cup (F_1 \cap I_2)$$

$$= I_2 \cap (F_1 \cup I_1).$$

We may now give the final form of the word-order algorithm adding the condition that $F_i = 1$:

Given a string of substituents S_1 , S_2 , ... S_n , with dictionary entries F_1 I_1 , F_2 I_2 , ... F_n I_n , for

any given substituent type, where S_1 is the last substituent in the string and S_n the first.

If $F_1 = 0$, $(S_1, S_2 \ldots)$ cannot form a valid bracket group.

If $F_1 = 1$,

Compute $I_0 = I_2 \cap (F_1 \cup I_1)$
and $F_0 = F_1 \cap F_2$. Then
if $I_0 = 1$, (S_1, S_2) is a valid bracket group under the word-order rule.

If $F_0 = 1$ and $I_0 = 0$, $(S_1, S_2,)$ is not a valid bracket group, but $(S_1, S_2, S_3,)$ might be valid.

If $I_0 = 0$ and $F_0 = 0$, S_1 and S_2 cannot belong together in a bracket group of that substituent type.

Unless I_0 and F_0 are both $= 0$,
Compute $I_0' = I_3 \cap (F_0 \cup I_0)$ and $F_0' = F_0 \cap F_3$ then
If $I_0' = 1$, (S_1, S_2, S_3) is a valid bracket group.
If $I_0' = 0$ and $F_0' = 1$, (S_1, S_2, S_3) is not a valid bracket group, but (S_1, S_2, S_3, S_4) now might be valid.

If $I_0' = 0$ and $F_0' = 0$, then S_1, S_2, S_3 cannot belong together in a bracket group of that substituent type.

Unless I_0' and F_0' are both $= 0$

Compute $I_o'' = I_4 \cap (F_o' \cup I_o')$

and $F_o'' = F_o' \cap F_4$.

Continue until for some set of substituents $S_1 , \ldots S_n$ a valid bracket is formed.

(c) The Concord Algorithm:

The form of the Concord Rule is such that a recursive algorithm is not strictly necessary. As we wish, however, to make a combined algorithm of all three rules, it is convenient to design the concord algorithm in this form. It can easily be verified that a recursive algorithm with

$$S_o = S_1 \cap S_2$$

and $P_o = P_1 \cap P_2$ (where 'S' here means 'being singular' and 'P' means 'being plural'), satisfies the concord rule we have given. Thus for any set of substituents $(A_1 \; A_2)$ if either S_o or P_o =1 then $(A_1 \; A_2)$ form a valid bracket group; and if both S_o and P_o are =0, then A_1 and A_2 cannot appear in a bracket group of that substituent type.

(d) The Combined Algorithm:

We now combine the three algorithms given before. First we note that the failure of any of the three causes the bracket group to be rejected. In other words

if g_0 and d_0 are both =0 or

if F_0 and I_0 are both =0 or

if P_0 and S_0 are both =0 or

if F_1 is =0, the bracket group fails.

Thus the bracket group is rejected if and only if the following holds:

$$\bar{F}_1 \cup (\bar{g}_0 \cap \bar{d}_0) \cup (\bar{F}_0 \cap \bar{I}_0) \cup (\bar{P}_0 \cap \bar{S}_0) = 1$$

or $F_1 \cap (g_0 \cup d_0) \cap (F_0 \cup I_0) \cap (P_0 \cup S_0) = 0$ .....(1) and further repetition of the algorithm to add further substituents will also fail. Thus this gives the stop rule.

For the bracket group to be valid as it stands, all three rules must be satisfied, that is we must have the following $g_0 = 1$

and $I_0 = 1$,

and either P_0 or $S_0 = 1$. The bracket group is thus valid if and only if the following holds:

$$g_0 \cap I_0 \cap (P_0 \cup S_0) = 1 \text{.....(2)}$$

In all other cases the bracket group is not yet valid, but may form part of a larger bracket group which will be valid, and the algorithm must be repeated to bring in the next substituent.

Substituting in (2) above for g_0, I_0, P_0, and S_0, we

have the following as the necessary and sufficient condition for the validity of a bracket group:

$$[(d_1 \cap g_2) \cup (g_1 \cap d_2)] \cap [I_2 \cap (F_1 \cup I_1)] \cap [(P_1 \cap P_2) \cup (S_1 \cap S_2)] = 1$$

or $$(d_1 \cup g_1) \cap (d_1 \cup d_2) \cap (g_2 \cup g_1) \cap (g_2 \cup d_2) \cap I_2 \cap (F_1 \cup I_1) \cap$$

$$\cap (P_1 \cup S_1) \cap (P_1 \cup S_2) \cap (P_2 \cup S_1) \cap (P_2 \cup S_2) = 1$$

and substituting in (1) above we have the following as the necessary and sufficient condition for the failure of a bracket group (omitting the special starting condition of $F_1 = 1$)

$$[(d_1 \cap g_2) \cup (g_1 \cap d_2) \cup (d_1 \cap d_2)] \cap [(F_1 \cap F_2) \cup (I_1 \cap I_2) \cup$$

$$\cup (F_1 \cap I_2)] \cap [(P_1 \cap P_2) \cup (S_1 \cap S_2)] = 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

SECTION 2

The Strategy of a Syntax Programme

We have given a definition of a valid bracket group, in terms of Dictionary information, and we have obtained a complete algorithm for computing whether or not any string of substituents will form a valid bracket group. We must now consider the types of strategy to be employed in recursively applying this algorithm to a sentence, so as to obtain a bracketting analysis of the sentence such that the complete sentence is exhibited as a substituent, or valid bracket group. We will illustrate the application of differing strategies to a common English sentence form:

((ab)((cd)(e(fgh))))

corresponding for example to the sentence 'The man was sitting on the park bench'.

We can first distinguish between two broad classes of strategy. The first is the Target strategy where we first attempt to find the largest, or outermost bracket groups, then split each of these into their inner groups, and so on. Using the example above, we would first look for (ab)(cdefgh), then, assuming this division to be correct, to split (cdefgh) into ((cd)(efgh)), and

then ((cd)(e(fgh))).

In the second strategy, we look first for the innermost bracket groups, and then combine them into successively larger groups until the whole sentence is contained in a single group. Thus we might first find the group (fgh), then combine this with the word 'e' to get (e(fgh)), find (cd) and combine it with (e(fgh)) to get ((cd)(e(fgh))) and then finally combine thi with (ab) to get the complete sentence.

If it were the case that the bracketting process was entirely unambiguous, that is, that in no case could a bracket group be formed which would not be incorporated in the final bracketting of the complete sentence, then sentence analysis would present no difficulties, and there would be little to choose between these two types of strategy. This would imply for example that (gh) or (ef) could never, in any context, be a valid bracket group.

The fact that this is not the case causes both types of strategy, if simply applied, to run into difficulties. The manner in which this breakdown occurs appears to be different in the two types of strategy, but is in fact basically the same. In the case of the Target strategy, the sentence is exhibited as a bracket group from the start,

and we look for the constituents of this bracket group. There will in practice be a number of different possible constituents, and each of these in turn must be broken down into its constituents, of which there may again be a number which are possible. Each assumption concerning the constituents of any particular bracket group leads to a tree of further bracket groups and further assumptions concerning their constituents, all of which appear, to the machine at least, to be different, and all of which have to be followed up. We cannot know which branch or branches of this tree will prove to be correct except by discovering those which eventually lead to failure, which they do by exhibiting a bracket group with no possible constituents. This will not necessarily happen until the level of the words in the sentence is reached.

In the case of the second strategy, on which the previous computer programme of the C.L.R.U., M.L. 136, was based, the breakdown occurs in an apparently very different way. By looking first for the innermost bracket groups, that is, groups formed from the sentence words, and then successively combining them into larger groups, we can be certain that any group formed is valid, and that its constituents are also valid. However, when attempting to form the groups we will in practice have a number of choices. Thus in the

example above, we might find that not only is (fgh) a valid group, and this leads to (e(fgh)), but also (ef) and (gh). If then ((ef)(gh)) is not a valid group, we can discard (ef) and (gh), but it may well occur that the latter group is also valid, and we might pursue it successfully up to the end of the sentence, only to find that the last bracket group containing the whole sentence cannot be formed. As in the case of the Target strategy, we again obtain a tree, each branch of which is formed by the different possibilities of combining a given set of substituents into valid bracket groups. And again, we may not be able to discard a branch until the analysis has been completed.

Given the existence of this problem, there are basically two sorts of tactics which can be employed - that of parallel computation and that of series computation. These apply equally to the two strategies described above. We must accept that we shall have to compute bracketting possibilities that will later prove to be unacceptable, and we must decide whether to compute these in parallel, keeping in the store all the possible brackettings obtained and following up all the branches of the tree at each stage, or whether to compute in series. In this latter case, we choose at each stage one only of the bracket possibilities,

and thus follow up only one branch at one time. If the procedure fails, we then track back up the branch, choose a different bracket possibility, and follow down a different branch.

In the previous C.L.R.U. Programme, we adopted serios computation. The small size of the then EDSAC II store made this method most convenient, and it also gave us the opportunity of studying the effects of a priority ordering on the bracketting possibilities. At every stage in the computation, when we are combining a set of substituents into a bracket group, we may have a choice of combination. We may choose any one on an arbitrary basis, or we may have built in an order of priority of choice which determines which combination is used first for further computation. This procedure is equivalent to the procedure used with target strategies of assigning different 'expectations' to certain types of bracket group, that is for example, of 'expecting' the first group of a statement to be a nominal group acting as the subject of the sentence. The effect of such an order of priorities is to minimize the amount of computation on a probabilistic basis. A priority rule is not absolute, a low priority group may in fact be the correct one for that particular sentence, but in general we can say that an acceptable

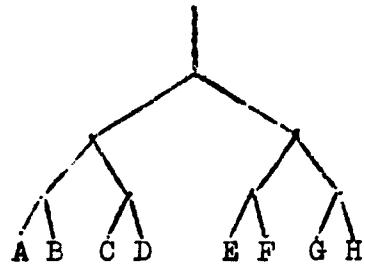
solution will be produced with less back-tracking than would be the case if the choice were arbitrary.

In a series method of computation, it is of course necessary to keep a record of all the choices which have been made, that is, of all the branches of the tree which have been followed up, so as to be sure that the same computation is not endlessly repeated. This was effected in the previous C.L.R.U. Programme by the device of an 80 bit 'Choice Tree' number. In this, all the possible paths of the computation were mapped on to a binary tree, in which each fork represented a binary choice. As soon as the choice was made, the corresponding bit in the choice tree number was set to '1', thus excluding that choice in the programme should that fork be revisited as a result of back-tracking. It was possible to observe the amount of back-tracking taking place, by printing out the 80 bits of the choice tree number. Experiments with this revealed a serious difficulty, which applies generally to any series method in this field. This trouble was manifested through the programme outputing a large number of apparently identical partial bracketings of a sentence, which could however be proved to have been each obtained by a different calculation.

Any series method for exhausting a set of choices can

be compared to the solution of a maze. We may solve a maze by a set procedure which prevents us taking the same path twice. We may then exhaust all possible paths. Thus when we reach a fork, if unmarked we mark it and take the left path, if already marked, we take the right path. If every blind alley has only one path leading to it, as is true of a real maze, then as we travel each path only once we also end up at each blind alley only once. If however there are more than one path to any given blind alley, then we are liable to arrive at that blind alley once for each path. Furthermore, as we have no way of knowing what set of paths lead to the same end-point (if we knew this the maze would be already solved), and we can only mark the paths, we have no way of preventing ourselves from reaching the same end-point a number of times. From the point of view of a Programme then, we have no way of preventing the repeated occurrence of partial bracketings which we already know will fail, and the programme will take a very great deal longer to come to its conclusion than would have been expected from the number of possible brackets which could be formed.

To illustrate this, consider the choice tree below:



The outcome points A,B,C- - - H represent bracketings or partial brackettings of the sentence. We can exhaust all the possible paths, and thus exhaust all the possible outcome points in a given order, say A, then B, C, up to H, and we will reach each point once only. It will in general be the case however, that several points will represent the same partial brackotting. Thus points A, C, F, and H might well all correspond to the same output, even though they are arrived at by different paths in the programme.

There appears to be little doubt that a syntax programme is of the form of a maze where any given outcome may be reached by a number of different paths, and there seems to be no reason to suppose that this difficulty would be any different if a target strategy were adopted.
(A.F. Parker-Rhodes is currently investigating a form of Target Strategy which may partly overcome this difficulty).

This then is an important disadvantage of the series method.

It is worth here examining one of the advantages of the series method. This is that it allows one to obtain a correct bracketting of a sentence without exhausting all the possibilities. In a parallel method, all the possible bracketings will be followed up, and the output of the programme will contain all those bracketings which survive to the end. In a series method, even with an arbitrary choice procedure, one complete output will usually appear before all the paths have been exhausted, and with an efficient priority order, an acceptable bracketting output may be achieved very quickly. We may then be content to stop and regard this output as the final result. We should here consider however, the distinction between a bracketting which is correct to the machine, that is, obeys all the rules of the programme, and a bracketting which would be acceptable to a person with a good knowledge of the language. That these two concepts of 'correctness' of a bracketting are not identical is shown by the number of machine correct, but obviously absurd, bracketings which can be produced.

It is clearly the aim of experiment to reduce the divergence between these two concepts of correctness as far as possible, and to do this with a minimum of dictionary information and computer routines. On doing tests on a

particular programme, with particular algorithms and dictionary information, we are concerned with comparing machine correct solutions with intuitively correct solutions and in testing the effect of further information in reducing the divergence between these. For this purpose, we are not only interested in the first machine correct solution produced, but in all possible machine correct solutions. Only in this case can we be certain, for any given sentence, that the apparent success of the programme was not due to some possibly accidental effect of a priority order, and that the successful result was not just one of a large number of absurd results, or on the other hand that in an unsuccessful result, the correct result would not have been obtained at the second try. Further, the amount of information obtained from any one test is greatly increased when all possible outputs are obtained, and it becomes less important to test such very large bodies of text.

Finally, the exhibition in the output of all the possible bracketings will enable us to test the discriminating power, as between these several bracketings, of other, not necessarily syntactical, rules. We could then test the power of semantic procedures applied after the syntactical analysis, without having to embed these procedures in the actual programme. For experimental purposes at any rate,

there then seem to be strong advantages in favour of a procedure for exhausting all bracketting possibilities, and in this case the series method is not acceptable.

The very considerable increase in the size of the core store of EDSAC II has now made it practicable for us to use a parallel computation method, which will provide as output all the possible machine correct brackettings of a sentence, and work is proceeding at present on a programme to do this, using the algorithms described in Section I. Before starting on this programme, we considered the possibility of using a binary bracketting parallel procedure, such as has been studied elsewhere. [Private communication from M. Kay of the Rand Corporation]. In this type of procedure, it is assumed that all brackets are binary - that they contain only two constituents. It is then possible to start at one end of a sentence, comparing the substituents in pairs and adding any binary bracket thus found to a list which is also taken in pairs with the remaining substituents. At any stage, a bracket either is or is not valid, as there are only two constituents, and any brackets which prove to be invalid are dropped from the list and have no further effect.

The C.L.R.U. Syntax Theory does not define a valid bracket as one that is necessarily binary, and such a

definition is incompatible with our definition. But, if such a rule as e.g., "seek to find only binary bracket groups", were added to our algorithm, this would, in certain cases, lead to difficulties, where (S_1, S_2, S_3) form a valid bracket under our algorithm but neither (S_1, S_2) nor (S_2, S_3) nor (S_1, S_3) form a valid bracket. If we do, nevertheless, attempt to add such a rule, the following disadvantages are apparent. First, there are a number of bracket groups occurring in language which do not fall easily into a binary form. A conjunct group ('boys and girls') is essentially ternary, and can only be compressed into a binary form by adopting some arbitrary conventions. The same is true of groups such as 'time after time', 'one by one'. A second objection is that a binary bracketting programme using parallel computation does not really get over the branching difficulty which occurs with series computation, as described above. One way of expressing this difficulty would be to say that it causes the programme to produce a number of effectively identical results, which are arrived at in different ways. A binary bracketting programme, when applied to language which is not in fact binarily bracketted, produces very much the same problem. To take an example, a simple nominal group,

'a dark blue hat'

We would wish to bracket this as '(a dark blue hat)' .
A binary programme will find all the following:

((a dark)(blue hat))
(a(dark(blue hat)))
(a((dark blue)hat))
((a(dark blue))hat)
(((a dark)blue)hat)

So far as the machine is concerned, these are all different. So far as we are concerned, the programme has made an entirely arbitrary distinction, which must be removed in some way or the number of machine correct bracketings produced will become unmanageably large. This appears difficult to accomplish without running directly into the branching problem which a parallel computation method is designed to avoid.

In effect, we must perform an additional computation which involves finding the longest bracket groups (those with the maximum number of substituents), which are equivalent to the binary brackets produced by the binary bracketting programme.

We thus considered it necessary to embark on a parallel computation programme which makes no restriction

on the number of substituents in a bracket group, and which can also exploit the computational advantages of the recursive algorithm given in Section I. This programme will use the second strategy described, that of finding the inner brackets first, and, as with the previous programme, will start at the end of the sentence. We start at the end of the sentence for the same reason that a programme using a target strategy starts at the beginning of a sentence - it is an empirical fact that in any language the normal bracketting structure is one in which the brackets cluster up at the end. Thus ((ab)(c(d(ef)))) is common, while (((((ab)c)d)(ef)) is uncommon. As Yngve (21) has pointed out, there appears to be good reason for this, in terms of the memory capacity required for a given depth of bracketting using a target strategy. In the programme strategy we are using, finding the inner brackets first, we can save storage space if we find a complete inner bracket group quickly, and it is thus more efficient to start at the end of a sentence.

The procedure of this programme, which determines the order in which substituents are compared by the algorithm and made into bracket groups, is unfortunately very much more complex than that which can be used in the case of binary brackets. In the latter case, any putative bracket

group can only have two members, and is unambiguously either valid or not valid. Neither of these conditions holds in the general case of non-binary brackets. The first clearly cannot be true, and the second is three valued. To go into the latter more deeply, it is true that by the algorithm we can say whether any given string of substituents S_1, S_2, S_3, ...S_n is or is not a valid bracket, taken by itself. For the purposes of obtaining a complete bracketting of a sentence, however, we need to know more than this. We need to know, given again a string of substituents S_1, S_2, ... S_n, first whether (S_1, S_2) is valid or not, and secondly whether the longer string $(S_1, S_2, \ldots S_x)$ <u>might</u> be valid so far as the information concerning S_1 and S_2 is concerned. As has been seen in the section on the algorithm, the algorithm gives us three possible cases. (i) The substituents cannot occur together in a valid bracket group. (ii) The substituents do not form a valid bracket group as they stand, but might form one when further substituents are added to the group (e.g. a governor). And (iii) The substituents form a valid bracket group, in which case they might also still form one when further substituents are added to the group.

In the place of the one list of binary brackets, we require two lists, one for brackets which are known to be

valid, and one for brackets which are potentially valid. On the other hand, these lists will not grow to the extent of the binary list, as the 'a dark blue hat' trouble described above cannot occur. The possible bracket groups in that case will be:

a dark (blue hat)
a (dark blue hat)
(a dark blue hat)

The possibility of, for example, (dark(blue hat)) is specifically excluded by the dictionary entry for a substituent type. An endocentric group may not be a member of a group of the same type. Thus (blue hat), as a nominal group, will not bracket with 'dark' in another nominal group, and (dark(blue hat)) cannot be exhibited as a nominal group, although (dark blue hat) can be so exhibited. We would also be able to deal straightforwardly with ternary groups such as conjunct groups.

An optimal search procedure using such lists has not yet been fully worked out, and the practical problems involved are considerable. There is no theoretical reason why such a procedure should not be found, for this is essentially an effective, finite problem.

The programme will be able to deal with any recursively defined algorithm giving rules of combination of substituents into bracket groups. Provision is being made for three different algorithms, each operating on the information contained in one 40 bit computer word. These will be used for the Governor-Dependent information, the Word-Order information, and the Concord information as described before, but other algorithms operating on other information could be substituted without much difficulty.

SECTION 3

The Algebra of Punched Cards

INFORMATION : QUESTION AND ANSWER :

As R.G. Collingwood (3) emphasized scientific knowledge may only be arrived at by asking questions. We cannot of course prove this, but it provides a definition of 'information' or 'content' ; at least it provides us with a definition that enables us to have computable information.

(1) If a question is answered (say by a dictionary-maker), i.e. if a numerical value (1 or 0 or Prob a/b) has been assigned to it, then the answer is well-formed, and the function to which the value has been assigned is well-defined and thus Turing-computable.

(2) If the answers to the questions are definite, e.g. the result of a football match - Win, Lose, Draw, i.e. if one of a finite set of values are assigned (e.g. 'Yes' or 'No'), then the answers may be treated as well-formed formulae in an n-valued logic, where n is finite; and in this case it may be reduced successively to the two-valued case. This is not in general true when the third value is "doubtful". Thus (3) Where the answers to the question are 'Yes/No', they constitute a two-valued

propositional calculus; each answer, conventionally written '1' or '0' will be a proposition, or a bit of information.

We shall here consider the third case in detail. First, because punched card equipment fundamentally produces only two states, for either it punches a hole or it does not; i.e. it performs only two operations, it punches a 'hole' or it punches a 'not-hole'. Second, because the properties of the circuits used in punched card equipment have been known, since Shannon (14) to be analogous to the Boolean algebra of propositions. Third, because the use of three-valued or n-valued logic as proposed by Lukasiewicz, (8), Reichenbach (13) or von Weizsacker (21), is not generally agreed to be valuable. Fourth, because in our syntactical investigations, we have, up to now, compelled 1 or 0 answers by framing the questions about words and substituents in terms of the possibility of the occurrence of a property (thus, e.g. we have not asked 'Does this word occur as a governor in a nominal group?) but 'Can it occur....?).

The Logical Algebra

The logic which we shall use to interpret punched card computation is elementary: i.e. it is the Boolean

algebra of propositions, or classes or properties (for these may be mapped isomorphically, one to each other). Originally formulated by George Boole, (1) and (2), it was adapted by C.S. Peirce (11) and E. Schröder (16) and expounded systematically by C. I. Lewis (7). It has always been known that a complete decision procedure for such a logic exists: we shall not be considering the Calculus of Predicates or any calculus involving unlimited quantification, since it has been known since Turing (18) and Church (4) that there exists no decision procedure for such a calculi, and therefore no complete computable punched card representation could exist, though Boole himself did give a representation of a calculus of predicates in terms of his algebra.

Though it is not essential to the expcsition we shall choose the propositional interpretation, and follow the computable rules suggested by D. Hilbert and B. Bernays (5) and D. Hilbert and W. Ackermann (6). We shall also use the closely connected Truth-table notation of E.L. Post (12) and L. Wittgenstein (20). We use the following notation

'p...q...r..' represent propositional variables whose values are the names of propositions.

'$p \cap q$' read 'p cap q' will represent 'and' i.e. logical conjunction or intersection. It is defined by the truthtable

$\cap$	p	q
1	1	1
0	0	1
0	1	0
0	0	0

'$p \cup q$' read 'p cup q' will represent 'or' i.e. logical disjunction or union. It is defined by the truthtable

$\cup$	p	q
1	1	1
1	0	1
1	1	0
0	0	0

$\bar{p}$ read 'not-p' will represent classical complementation or negation. It is defined by the truthtable

$-$	p
0	1
1	0

'$\emptyset = I$' read '$\emptyset$ is a tautology' will represent the assertion that the formula is true no matter what the values of the constituent variables

It is defined by the truthtable

$\emptyset = 1$	p	q
1	1	1
1	0	1
1	1	0
1	0	0

' $\emptyset = 0$ ' read ' $\emptyset$ is a contradiction' will represent the assertion that the formula is false no matter what the values of the constituent variables.

It is defined by the truthtable

$\emptyset = 0$	p	q
0	1	1
0	0	1
0	1	0
0	0	0

It has been known for a long time that these functions are sufficient to define any other truth-function of elements (of the values / of the elementary propositions given by the bits), of which in the case of two valued logic there are 2^{2^n}, where n is the number of elements. E.g. 'p $\rightarrow$ q' may be defined as $\bar{p} \cup q$ ('material implication'), and 'Equivalence' as $p \rightleftharpoons q$, thus permitting algebraic computation, because equations can be written in the notation, and Boole's original addition operation, ■ +, may be defined as $\bar{p} \cap q \cup p \cap \bar{q}$.

They are also sufficient to set up this algebra axiomatically, which is uninteresting for computation.

We shall use the following properties of these functions

(1) $\cup$ and $\cap$ are idempotent, commutative, associative, and distributive over each other, i.e. the following equivalences hold:

(a) $p = (p \cap p) = (p \cup p)$

(b) $(p \cap q) = (q \cap p)$, and $(p \cup q) = (q \cup p)$

(c) $p \cap (q \cap r) = (p \cap q) \cap r$ and $p \cup (q \cup r) = (p \cup q) \cup r$.

(d) $p \cup (q \cap r) = (p \cup q) \cap (p \cup r)$ and $p \cap (q \cup r) = (p \cap q) \cup (q \cap r)$.

(2) $-$ is classical i.e. the following equivalences hold:

(a) $p = \bar{\bar{p}}$

(b) $\overline{p \cap q} = \bar{p} \cup \bar{q}$

(c) $\overline{p \cup q} = \bar{p} \cap \bar{q}$

(d) $\bar{I} = O$

(e) $\bar{O} = I$

(3) The negation of any function $\phi(p \ldots q)$ may be constructed by replacing each constituent bit by its negation, each occurrence of $\cup$ by $\cap$, each occurrence of $\cap$ by $\cup$, and each occurrence of I by O, and O by I.

(4) $\phi \cap I = \phi$, $\phi \cap 0 = 0$

$\phi \cup I = I$, $\phi \cup 0 = \phi$

(5) Where an equivalence $\phi = \psi$ is shown to be a tautology, another equivalence which is its dual, may be obtained (if ϕ and ψ consist only of elements and their negations connected by $\cap$ and $\cup$), simply by interchanging $\cap$ and $\cup$ throughout the formula. Thus if

$\phi \left[(p..q) = \psi (p..q)\right] = I$ then

$\phi' \left[(p..q) = \psi' (p..q)\right] = I$ may be constructed such that

$\phi' (p...q) = \psi' (p..q)$ differs from

$\phi (p..q) = \psi (p..q)$

only in that $\cup$ is replaced by $\cap$ and $\cap$ is replaced by $\cup$.

(6) From (3) and (5) it follows that for any formula $\phi = I$ there will be another formula $\phi' = 0$ constructed by replacing each bit by its negation, and $\cap$ by $\cup$ and $\cup$ by $\cap$. Thus one may either compute whether a formula is $= I$, or by using this rule, arrive at an equation, $\phi' = 0$ to which there is always a solution.

(7) For any formula $\phi (p...q)$ there exists a representation in the so-called Normal Forms. The Conjunctive Normal Form is a Conjunction of brackets, where each

bracket contains a Disjunction in which each element is an elementary proposition or a negated elementary proposition. The Disjunctive Normal Form is a Disjunction of brackets where each bracket contains a conjunction in which each element is an elementary proposition or a negated elementary proposition.

(8) Using $p=p\cap(q\cup\bar{q})$, the Expansion or Development of p with respect to q is $(p\cap q)\cup(p\cap\bar{q})$. Hence any function of elements $\phi(p...q)$ may also be treated as a function of any other element n, though originally there is no information concerning n.

(9) The negative of a formula may be constructed by forming the Expansion of I with respect to all the elements that appear in the expression For $I = p\cup\bar{p}$

Thus $I = (p\cup\bar{p})\cap(q\cup\bar{q})$

$$=(p\cup\bar{p})\cap(q\cup\bar{q})\cap(r\cup\bar{r})\ldots\ldots\ldots\ldots(n\cup\bar{n})$$

Then the negative or any fully expanded expression is the remainder of the expansion of one. This may be written, following Boole, (1) $(1-\phi)$. This enables us to introduce $-$ subtraction (the inverse of $+$ or Boole's addition operation). We have thus got an arithmetic definable in the logic, and can compute any natural number in it.

(10) There is a representation for any formula in the so-called 'Ausgezeichnete Normalform', the Exceptional Normal Form. This is constructed by developing the formula $(p \cap \bar{p}) \cup (q \cap \bar{q}) \ldots\ldots (n \cap \bar{n})$, in accordance with the distributive law, where n consists of all the elements required. This will produce an expression in the Conjunctive Normal Form, and each bracketted part of this expression will be equivalent to a distinct truth-function, and one set of these brackets will be equivalent to the required formula (except the degenerate case of the tautology, which is represented by the zero set of such brackets). In order to produce the Exceptional Normal Form of an expression it must first be reduced to the Conjunctive Normal Form. Assuming that the elements in question are p.....n), we drop all brackets of the form $(p \cup \bar{p})$ and write p for $p \cup p$ throughout. Then each remaining bracket will be a disjunction of distinct elements of the sequence p...n. If, in any bracket, both q and $\bar{q}$, say, are lacking, we may extend the disjunction by the element $(q \cap \bar{q})$ (for this =0) and apply the distributive law until each bracket contains, for either each element p, that element or its negation. (Since $\cap$ is commutative, distinct orderings are equivalent and only one need be written.

Because this representation corresponds to the truth-

table notation, two formulae which have the same representation in this Normal Form are truth-equivalent: this enables us to give a unique representation of any formula. We may arrive at the simplest form of a formula by giving it this representation and applying the Elimination Rule stated in Hilbert and Ackermann (6):

$$(\phi \cup \psi) \cap (\bar{\phi} \cup \psi) = (\phi \cap \bar{\phi}) \cup \psi = \psi$$

(11) In order to enable us to write down in a mechanical fashion, the algebraic formula corresponding to a truthtable, we assign truth-values in the following manner, which is both fixed and exhaustive: 1010 for the first element, 1100 for the second and so on.

Since punched card equipment is designed to record information, it is obvious that there will be a representation of this algebra in the cards and appropriately designed equipment. The equipment we consider here is the Keypunch, The Duplicating punch, the Reproducer Punch, the Sorter, and the Collator, and some actual and possible modifications of these.

The Card:

The card to be considered is the I.C.T. 407-353., containing 12 rows and 80 columns, i.e. there are 960

mechanically distinct spaces. On each of these a hole may or may not be punched (here the hole is exclusive to the "not-hole," as "Yes" is exclusive to "No" ; thus conventionally a hole is taken to represent "1" and a blank space is taken to represent "0".) It is worth noting that the opposite interpretation is possible and some interesting results follow from the possibility of reversing the interpretation; we shall return to this.

The card is a two-dimensional space which is divided conventionally into 12 rows and 80 columns on to which information is punched. However, information appearing on any card can be transferred mechanically from any set of columns to any other set. Thus the order of the information in the columns, which may easily be taken to be significant, may be changed or shuffled, by the use of the Reproducer punch. Since any substitution of columns is possible, the card may be conceived of as bent round a cylinder, and <u>rotation</u> of <u>column</u> and <u>column information</u> is possible.

It is not, in general, possible to transfer information from row to row in the same way, but to some extent this may be overcome by the <u>rotations</u> of the <u>card</u>, which we consider here for the first time.

The single card, which is an essential element in our

considerations, is capable of two independent rotations, i.e. any other rotation of the card is a combination of these two (but not of course the mapping of column information on to other columns referred to above). These rotations are (1) About the small diameter of the card, which we call the Right-Left rotation, and (2) About the large diameter of the card, which we call the Up-Down rotation. If we are prepared to sacrifice some of the information-capacity of the card by using only the 480 bits on the Left-hand side of the card (Columns 1-40), then the Right-left rotation makes possible comparison of information on the Right-hand side of the card with information on the Left-hand half. Similarly, if we are prepared to sacrifice information carrying capacity on the Down-side half of the card, the Up-Down rotation makes possible comparison of information on the Up 480 bit half with information on the Down 480 bit half. And if we are prepared to sacrifice information capacity still further, we may divide the card into quadrants of 240 bits each. The two rotations then make possible comparison of information on any part of the card with information on any other part of the card. These operations are conveniently performed on the Duplicating punch, modified only to permit two cards to be inserted simultaneously into the reader, but which operates fundamentally on single cards or a pair, not a pack.

Another extension of the range of computation is achieved by considering the three-dimensional array of a set of cards placed one on the other, or a pack. This may be arranged in any order. The Reproducer Punch, the Sorter, and the Collator are constructed to compute over a pack.

The Key-Punch:

The keypunch has only one basic operation - and this is true of all punched card equipment - it punches a hole in a given space. This operation is asymmetrical, i.e. a blank space or a 'not-hole' is immediately converted into a 'hole', (thus a '0' is mechanically converted into a '1', but without further change in the equipment, a 'hole' cannot be converted into a 'not-hole', thus a '1' cannot be converted into an '0'.) It is immediately obvious that this statement may be changed symmetrically, by taking the 'hole' to mean '0' and the 'not-hole' to mean '1' but the asymmetry still remains and a reversal of interpretation may not be done simultaneously. From this it immediately follows that there does not exist, in the punched card algebra, a general mechanical definition of negation or complementation as defined in our truth-table. There are important consequences to this and we shall return to the whole question of computing the negation of a formula.

The Detailed Representation of this Algebra on Punched Cards:

A hole represents the value '1' given to a variable, say 'p'. A blank space or 'not-hole' represents the value '1' assigned to a variable, say '$\bar{p}$' ; it thus represents the value '0' assigne to 'p'.

The card itself represents a function $\phi\ (p_1, p_2, \ldots p_{960})$ of the values of the elements $p_1, \ldots p_{960}$, or the bits of information. Thus if, for the sake of simplicity, we only consider the case of two elements 'p' and 'q' , the number of cards necessary to represent all distinct truth-functions of these elements will be 16. Of these one card will be blank, and this will represent =0 and one will have four holes ▪ (=1). The complete punched card universe of information is a card with 960 holes on it (this may be multiplied by taking packs of cards, with a certain inconvenience in considering the rules for reading the cards): but the universe with respect to the n elements of information under consideration is less: it is 2^n.

Since any problem in this logic may be formulated as asking whether a function $\phi = 1$ or =0, or whether some function of functions of elements (and the values of these functions are themselves =1 or =0 is =1 or =0), any problem which consists of computing the value of a function

of functions, may be represented by a pack of cards and the solution will be an output card, which is =1 or =0. Because all cards may be given a complement-card, by constructing one in accordance with the punching rule described below, and the computation of the equivalence of two functions is always reducible to an equation [.....] = 0, there are four remaining problems that arise: (1) to compute $\phi \cup \psi$, or the union of the information of a pack of cards; (2) to computethe conjunction or the intersection of the information on a pack of cards or the formula $\phi \cap \psi$.; (3) to compute (a) whether a particular card, representing at least an intersection of properties, F,G.H, exists or does not exist inma pack, and (b) whether a card <u>identical</u> to a particular card, i.e. representing only F,G,H, exists or does not exist in a pack; (4) to compute the complement or the negation of a given function or card, i.e. to compute $\bar{\phi}$.

There are general mechanical solutions to the first three problems, and though our punched card logic contains no operation of negation (such as would be provided, say, by a conditional switch which would reproduce from a card ϕ its complement, $\bar{\phi}$ by punching a hole where there was a 'not-hole' and a 'not-hole' where there was a 'hole') we have a general solution to the fourth problem. Although

we cannot in general perform algebraic computation involving negation, we can always compute the actual values of any well-formed formula, even when it involves negation <u>essentially</u>, if we are given the values required to do the positive computation.

(1) The general solution to (1) is available as soon as it is seen that the operation of duplicating n cards (for the sake of simplicity we consider only two), when performed successively on to the same output card, will produce the disjunction or the union of all the bits or the values of the elementary propositions on both cards. Thus we will have ψ with all the holes there are on ϕ as well as all the holes there are on ψ. This provides a general mechanical definition of $\phi \cup \psi$ and by the Law of Duality of

$$\bar{\phi} \cap \bar{\psi} .$$

(2) The general solution to (2) is similarly provided by the basic punching operation. If two cards are placed on each other in the reader (as in a Duplicating Punch modified to read two cards simultaneously), the holes on the output card ψ will be those that are both on ϕ and on ψ. In other words we can provide a general mechanical definition of $\phi \cap \psi$. And by the Law of

Duality of $(\bar{\emptyset} \cup \bar{\emptyset})$. The Dual forms are important because one may either reverse the interpretation of 1 and 0 or punch the complement of the information on the card.

(3) The general solution to this problem is provided by the solution to the other logical problems. We will reserve detailed consideration of this problem until we describe some of the possible operations of the Sorter, the Collator and the Reproducer: generally speaking this is the problem of selecting a function i.e. computing whether in a pack there is a card whose function has a certain relation to a pattern-card or function.

(4) The general solution to the problem of negation must be sought in the logic itself. It is clear that the existence of $\cup$ and $\cap$ and any function of these, suffices to construct a non-complemented distributive lattice or a 'positive logic' in the sense of Hilbert and Bernays (5) : it is therefore impossible to construct all the formulae which are permitted in a complete logical algebra. And both of the rules we have given, e.g. (3) and (9) of the properties of the algebra, which are inddpendent of each other, use the operation of negation, and it would appear that any computation of the value of a formula whose representation in the Exceptional Normal Form essentially involves

a negation sign cannot be done without the addition of a circuit or conditional switch, that will reproduce a card under the symmetrical condition that a hole is replaced by a not-hole and a not-hole by a hole. Certainly it would be very convenient to have such a conditional transformation, but it is not necessary, since we may instead double the number of elements in our universe by assigning to each '1' a '0' and to each '0' a '1'. We widen our algebra by treating all negations as if they were positive elements: to each of the original elements there corresponds a new 'positive' element whose relation to the original one is exactly that provided by a negation operation. In connection with the interpretation in terms of question and answer it is worth noting that 'Yes, p is not true' is truth-equivalent to, (is the same as), 'Yes, $\bar{p}$ is true', or '$\bar{p}$' i.e. 'No'.

We may construct the complement of a given card - or any element of which a card is a function -, by the following punching rule: wherever there is a '1' on the card, punch an '0', which we may call its 'positive negation', on the other card, i.e. do not punch in that space: wherever there is an '0' on the card, then punch a '1', i.e. a hole on the other card. It is clear that the intersection of these two cards representing functions ϕ and ψ

is =0, therefore $\not\emptyset$ is = $\not\emptyset'$. Considerable subtlety may be introduced here by considering the Up-Down partition of the card into two halves of 480 bits each as constructing two cards of which the Up and Down may be seen to be complements if the same punching rule is followed. A card $\not\emptyset$ will then contain two sections: Rows Y, X, and 1-4 will contain the information and Rows 5-12 will contain the complement of this information. Of course such coding will not permit the original card also to be treated as one function of elements i.e. $\not\emptyset$ (Up-Down) is not a function in our Boolean Algebra where the Up section represents say $\not\emptyset$ and the Down section represents its negation - $\bar{\not\emptyset}$. For any operation of union or intersection on the total card will produce a card $\not\emptyset$ (Up-Down) in which the Up and Down sections are <u>not</u> complements.

We now note the fact that any function $\not\emptyset$ (p.....n) of n elements, has a Normal Form representation where the bracketting in the formula is of only one level. We may thus have

$(p \cap q \cap \bar{r}) \cup (\bar{p} \cap q \cap s) \cup (p \cap \bar{q} \cap r)....$ and not $(p \cap q \cap \bar{r}) \cup [(p \cap \bar{q}) \cap \bar{s}]$.

The negation sign occurs only over the elements. Let $\not\emptyset$ (p...n) be such a function in the Normal Form with n

elements in a complete logic that includes negation. We define m new elements (where $m \leqslant n$) called p', q', r' etc., by the rule $p' = \bar{p}$, $q' = \bar{q}$, $r' = \bar{r}$, ... $m' = \bar{m}$. For every occurrence of $\bar{p}$, $\bar{q}$, $\bar{m}$, in ϕ substitute p', q', ...m'. We now have a new function $\phi'(p_1\ p_1',\ q_1\ q_1'\ r_1\ r_1'\ \ldots)$ with (n+m) elements which is also in the Normal Form, but is expressed in a positive logic. Clearly ϕ' and ϕ are truth-equivalent and as ϕ' is expressed in a positive logic, ϕ' and hence ϕ are computable on positive punched card machinery. Thus, given any function $\phi(n)$ in a complete logic, there is a fixed rule for converting it into an equivalent function of not more than 2n elements, in a positive logic.

In practice, this procedure does not cause any inconvenience, provided that the necessity for it is realised from the start, when the data to be processed is first punched on to cards. The standard punching convention is to punch a hole for a '1' (for a true proposition) and to do nothing, (i.e. push the space bar) for an '0'. If the computation to be performed on the data will require negation, it is essential to effect the substitution of $p' = p$ at the input stage, by punching a hole in one position for a '1' and a hole in a different position for an '0'. We might, e.g. punch a hole in row 1 for a '1' and a hole

in row 6 for an '0', thus allowing us to use the row transposition from Y-3 to 9-4, effected by performing the Up-Down rotation of the card, to take union or intersection with the element p, or its complement $\bar{p} = p'$. It is true that half the capacity of the card is lost, but this punching rule does not in fact increase the number of key depressions when punching the data.

We wish to emphasize once again that our solution of the problem does not suggest that a positive logic is complete in the sense in which a logic containing negation is complete; (here "a logic containing negation" is meant to include a logic containing operations such as Sheffer's (15) stroke-functions, $\uparrow$ $\downarrow$, or Boole's exclusive or, $+$, in terms of which a negation function is easily definable). All we assert is that because there is a Normal Form representation for any formula such that negation signs appear only over the elements, the value of a formula containing negations essentially may nevertheless be computed by doubling the number of elements in the algebra; the necessity for computing the negation of a formula is removed, by having the complements of the elements and computing positive functions of these elements and their complements.

The Duplicating Punch : Modified to accept a pair of cards in the reader, is a complete positive logic. We use the Right-Left card rotation. We can compute

(1) $\phi \cap \psi$ in one machine operation
(2) $\phi \cup \psi$ in one machine operation
and (3) $(\phi \wedge \psi) \cup (\phi' \wedge \psi')$ in two machine operations.

The Sorter:

This machine, which will reverse the order of a pack of cards, will solve the problem of selecting a card which contains at least a certain intersection of properties, from a pack. Given enough time, it may even select a card identical to a given function or card. It is the most primitive machine for extracting information.

The Collator: Ignoring the arithmetical properties of this machine, this machine can be treated as a more efficient sorter or selector. It will select from a pack, all those cards identical to a given master-card where the card consists of not more than 32 columns, each of which contains not more than one bit per column.

The Reproducer Punch:

This machine is fundamentally intended to duplicate either a large number of cards off a single card or to

reproduce packs of cards. We shall show that it can be regarded as a complete logic, and thus also as a complete information selection machine, far more efficient than the sorter, (which depends on asking Yes/No questions, column by column).

In terms of the punched card logic, the operation of duplication can compute the disjunction or union of the information in a pack of cards, by transferring the information on the first card on to the second, which then represents, say, $\phi \cup \psi$. This output card is then transferred on to the third card and so on to the end of the pack. In other words where n is the number of cards this machine will compute $\phi \cup \psi \cup \chi \cup ..n$; by the Law of Duality this is simultaneously a computation of the formula $\bar{\phi} \cap \bar{\psi} \cap \bar{\chi} \cap \bar{n}$. If we either have our information punched to include the complement cards of all cards, or preferably we add a negation or a conditional switch, we have a machine which will do all the necessary computation of our logical algebra.

The machine as it stands will also compute the unions or disjunctions of pairs of cards taken in order from two distinct packs; if the two packs in question represent the functions ϕ (..n) and ψ (...m) it will compute the sequence of unions $\phi^1 \cup \psi^1$, $\phi^2 \cup \psi^2$, $\phi^n \cup \phi^n$, and the

sequence of intersections $\bar{\phi}^1 \cap \bar{\phi}^1$, $\bar{\phi}^2 \cap \bar{\phi}^2$, $\bar{\phi}^n \cap \bar{\phi}^n$.

These two distinct machine properties depend on the fact that column information may be transferred to other columns or to the same columns.

In order to regard this machine as a selector, and thus as a more efficient sorter, we will take problem (3) and give a solution. (a) If we wish to extract from a pack of cards, all those which include, i.e. contain <u>at least a particular intersection</u> of properties, say F, G, and H, this is to extract from the pack all those cards for which $F \cap G \cap H = 1$, i.e. for which $\bar{F} \cup \bar{G} \cup \bar{H} = 0$.

The most economical solution is not to compute this union from card to card but from column to column within a card by plugging the columns together. In this case we wish to compute $\bar{F} \cup \bar{G} \cup \bar{H}$ where $\bar{F}$, $\bar{G}$, $\bar{H}$, are all on the same card. Assuming that the information has been singly punched and that $\bar{F}$ is in column 1, $\bar{G}$ is in column 5 and $\bar{H}$ in column 7, all, say, in Row 1, We may then plug the Reading brushes Columns 1, 5 and 7 together on to say, Punch Magnet Column 1, We reproduce the pack on to a blank pack, at the same time reproducing an identification number if required. The result is to obtain a second pack of cards, in one to one correspondence with our original

data pack, for which any card with a <u>not-hole in Column 1</u> corresponds to a data card with $\bar{F} \cup \bar{G} \cup \bar{H} = 0$ or $F \cap G \cap H = 1$. Thus each card with the desired property will be identified by a corresponding card with a not-hole in Column 1.

Alternatively by using the automatic error stop facility on the machine, we may cause the machine to stop whenever $\bar{F} \cup \bar{G} \cup \bar{H} = 0$ and then withdraw the card from the hopper. This may be done by plugging the columns to one side of a comparing relay, and the other side of that relay to a '1' emitter. An '0' output will then appear as an error to the machine.

It is worth noting that we may compute up to 80 functions of this type simultaneously: for we can compute, say, $\bar{F} \cup \bar{G} \cup \bar{H} = 0$ on column 1, $\bar{F} \cup \bar{G} = 0$ on column 2, $\bar{G} \cup \bar{H} \cup \bar{J} = 0$ on column 3 and so on.

We may also use a similar procedure to solve problem (3)(b), i.e. to find all cards which have <u>only</u> the properties $F \cap G \cap H$. If the total set of properties is A,B,C,D,E,F,G,H, the function we require to compute is

$$\bar{A} \cap \bar{B} \cap \bar{C} \cap \bar{D} \cap \bar{E} \cap F \cap G \cap H = 1 \quad \text{or}$$

$$A \cup B \cup C \cup D \cup E \cup \bar{F} \cup \bar{G} \cup \bar{H} = 0 .$$

Here it is necessary to have the information concerning

these properties : punched both as p and as $\bar{p}$, as e.g. in row 1 and $\bar{p}$ in row 6. After reproducing, our correspondence cards will have a not-hole in row 6 for all those cards with $\bar{F} \cup \bar{G} \cup \bar{H} = 0$ and a not-hole in row 1 for all those cards with $\bar{A} \cup \bar{B} \cup \bar{C} \cup \bar{D} \cup \bar{E} = 0$. We can then extract all those cards for which the correspondence cards have a not-hole in both row 1 and row 6.

It is clear that any problem of retrieving information can be coded so that the answer will consist in producing cards which have a specifiable relation to a given function of that information: usually "Are there any cards satisfying a given function of the properties"? or "How many are there"?. Certainly in the case of stock retrieval, or the selection of personnel, or the selection of a card containing the birth record of a particular individual - and these are three problems which punched card machinery is being used to solve at present - the usual machine operation is performed by sorting: this is not only slow but highly uneconomic. The Reproducing punch, if the information be punched to include complements, or modified as suggested, is obviously very much faster and more economic.

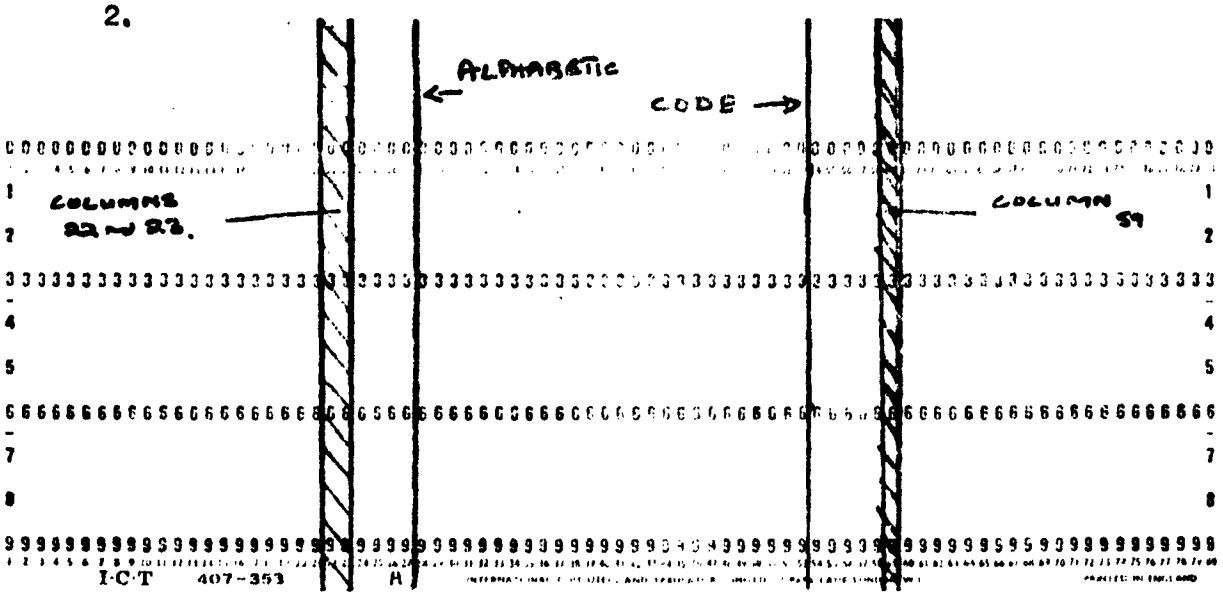
BRACKETTING WITH PUNCHED-CARD EQUIPMENT

1. The punched-card operations for syntactical sentence bracketting described here have their theoretical foundation in the work of A.F. Parker-Rhodes and their immediate justification in the algorithms derived by D.S. Linney and R. McKinnon Wood. The work was done on a Hollerith duplicating punch using two packs of cards punched with Governor-Dependent and Word-order information respectively for each word of the sentences examined. The former was obtained from the Governor-Dependent information provided for the words of the Longmans-Green Defining Vocabulary by Y.A. Wilks and E. Rigby, the latter was provided by D.S. Linney. The sentences themselves were taken from a Longmans-Green elementary English story-book ("Tales from the East") based on the Vocabulary. This paper describes attempts to obtain one satisfactory bracketting of each sentence. For this bracketting, i.e. this particular distribution of parenthesis signs, it is a necessary, but not a sufficient, condition for the formation of any particular bracket that the set of contiguous substituents under consideration can form a valid bracket. That is to say, all the attempts, starting at the right-hand end of the sentence, aim to locate the longest possible brackets.

The results are used to compare two "strategies"; one, the straightforward performance of the Governor-dependent and Word-order algorithms simultaneously for each pair of substituents considered ("in parallel") and, two, a strategy using G.D. in a normal and word-order information in a restricted form, together with certain other rules, in an endeavour to avoid some of the card storage problems arising with the other methods.

The latter method is outlined and accompanied by a partial formalization in terms of a flow-chart and a "manual program" in an adapted form of the Parker-Rhodes Algorithmic Notation (q.v.). This is preceded by (2) an outline of the forms of presentation of the syntactical information on the cards, and of the performance of the algorithms with the cards so punched; (3) a description of the notation used to display the bracketting process, and (4) a description of the preliminary operation of checking the dictionary. This last consisted in providing a satisfactory intuitive bracketting of each sentence and then operating with Governor-dependent and word-order cards in such a way as to ensure that the information punched on them at no stage exluded this bracketting in its correct substituent type. Were this not done a mistake in the compilation of the dictionary entry for a word could make all operations with that word useless.

2.



The centre portion of columns (28 - 53) was allotted to the punched code for the name of the word. The columns (2 - 27) on the left-hand side were divided into adjacent pairs from the left, i.e. 2 and 3, 4 and 5 etc. and one such pair was taken to represent the behaviour of the word in one of the 13 substituent types considered for English, i.e. the syntactical behaviour of the word (as Governor or dependent in the one case, as initial or final in order in the other) in substituent type 4 is recorded in columns 10 and 11, the types being considered in the order; conjunction, one, two.....twelve.

For both types of information there are four possible

entries in binary arithmetic for a word in any given substituent type; 00, 01, 10, 11. The first digit of the 2 digit figure is recorded with a hole in the numerically lower of the two columns, the second digit is similarly recorded in the next column and the column on the right-hand side whose number is (8_2 - number of the relevant column on the left-hand side of the card) - this is the punching procedure for Governor-dependent information: the Word-order information has its second digit punched only in the column on the right-hand side of the card. In each case if the digit is a "1" a hole is punched in row 1, if it is a "0" a hole is punched in row 6.

It will be evident that we have the possibility of two "dimensions of comparison" between cards; upper-lower and right-left of the card faces. D.S. Linney and R. McKinnon Wood have investigated the potentialities of computing logical sums and products by the method, and have shown that the Boolean algorithms for both Governor-dependent and Word-order algorithms can be given for a pair of substituents by 3 and 4 card operations each, using only the right-left hand reversibility; viz. for a pair of substituents A, M occurring in that order in a sentence.

i) Governor-dependent algorithm

1. Take intersection of left-hand side card (A) and left-hand side of card (M) backwards on left-hand side of card (C)

2. Take intersection of left-hand side of card (M) and left-hand side of card (A) backwards on left-hand side of card (C)

3. Take intersection of left-hand side of card (A) backwards and left-hand side of card (M) backwards on left-hand side of card (C) backwards.

Note that since the same Part of the same output card is used for 2 and 3 we have in effect produced a union of intersections.

ii) Word-order algorithm

1. Take union of l.h.s. of card (A) and l.h.s. of card (M) on l.h.s. of card (C)

2. " union of l.h.s. of card (A) backwards of card (M) on l.h.s. backwards

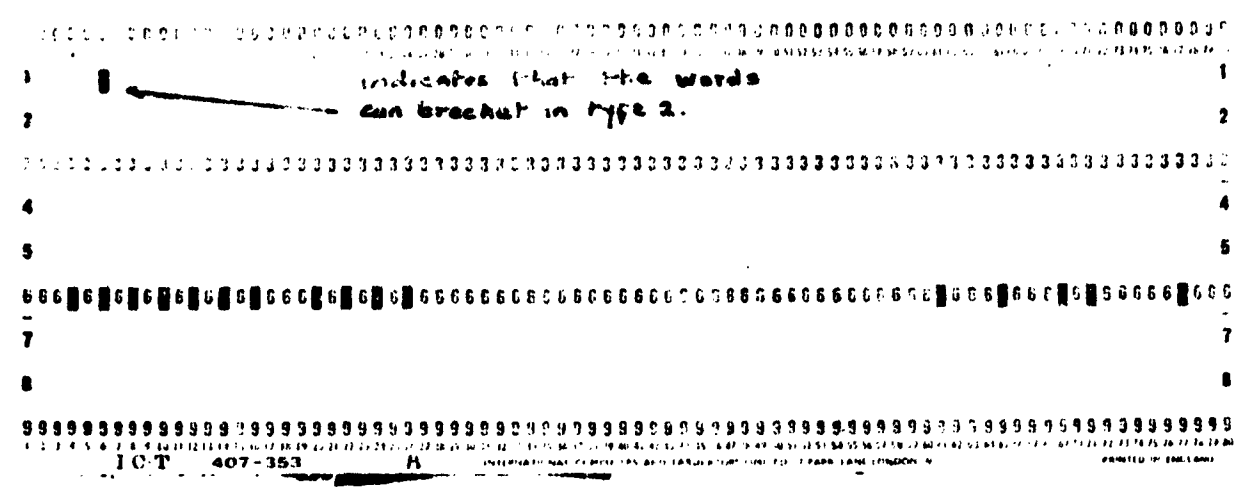
3. " union of l.h.s. of card (A) backwards of card (M) backwards, backwards.

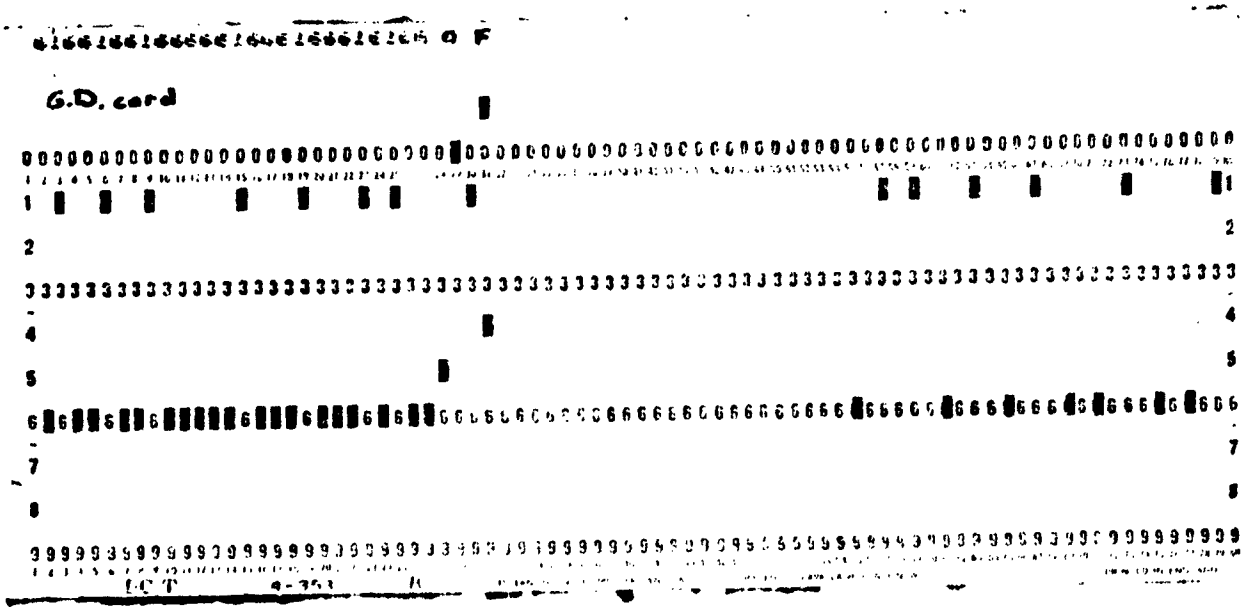
In each case the output card C indicates the syntactical properties of the combination of A and M, its interpretation is given on the reproductions of cards (see pp. 6-7).

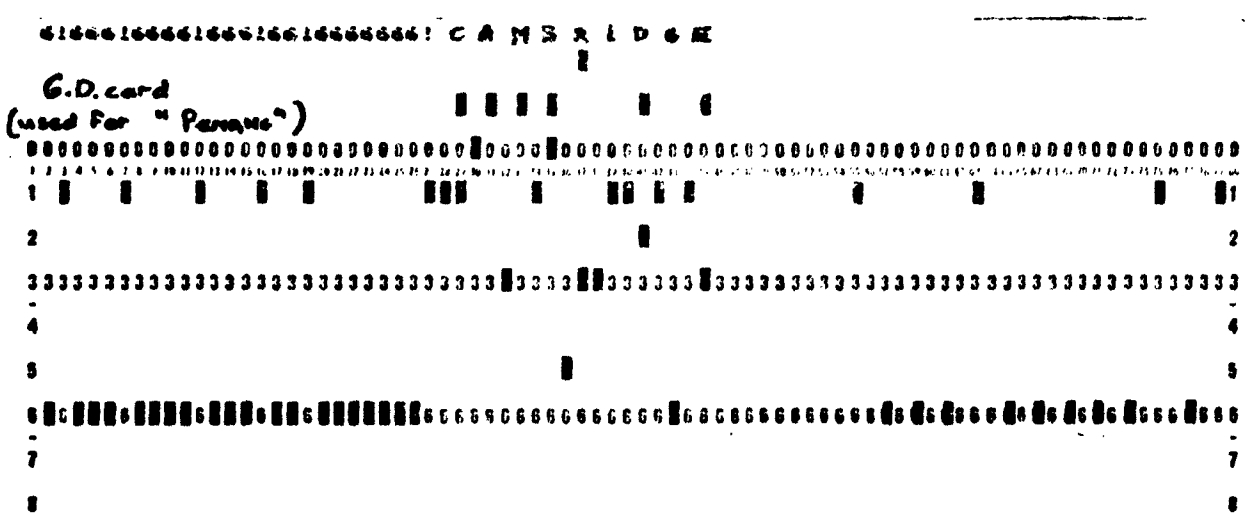
The algorithms are recursive and in each case the output card can be used as a substituent to determine the properties of the group A, M and the next word occurring in reverse sentence order.

The participation class entries for each of the 13 substituent types for English are recorded separately for Governor-dependent (GD) and Word-order (WO) information on what are called 'Master-cards', indicated by type and variety of information thus M(GD)7 ≡ "the card bearing the G-D participation class entry for any substituent of type 7".

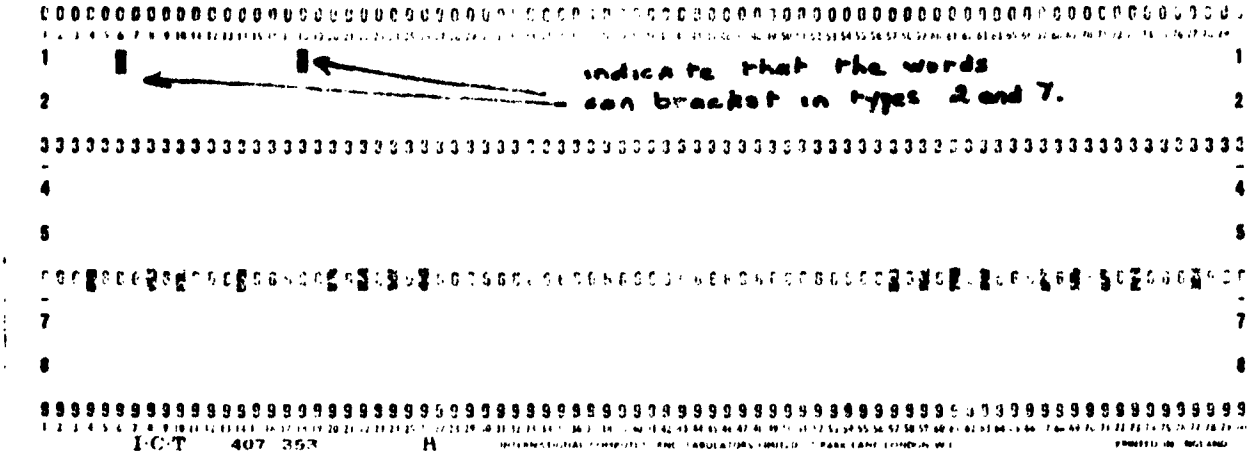
G.D. output card for "of Penang"





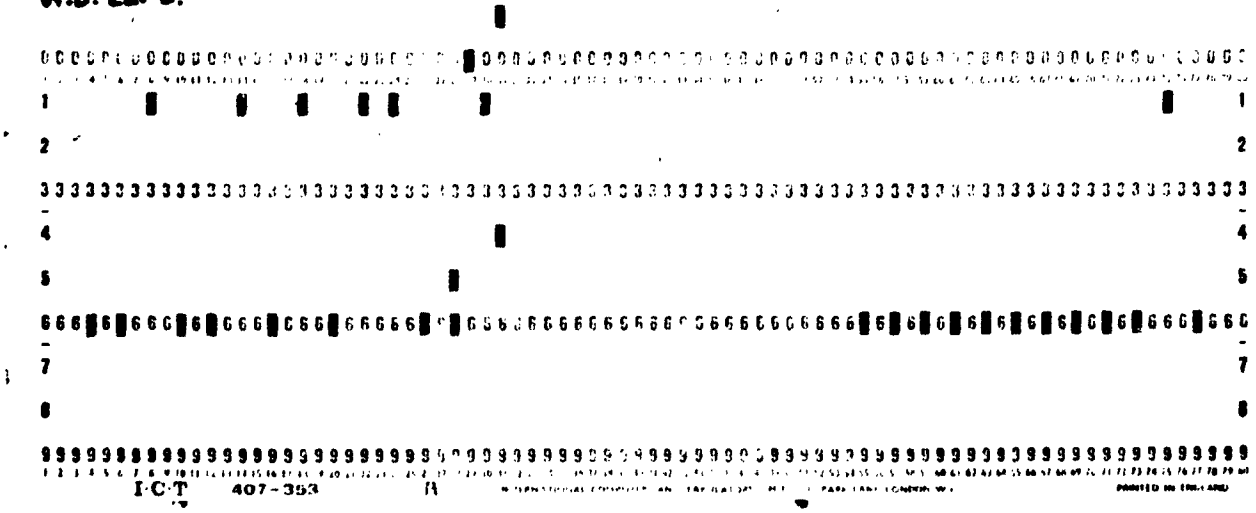


W.O. output card for "of Pennine"



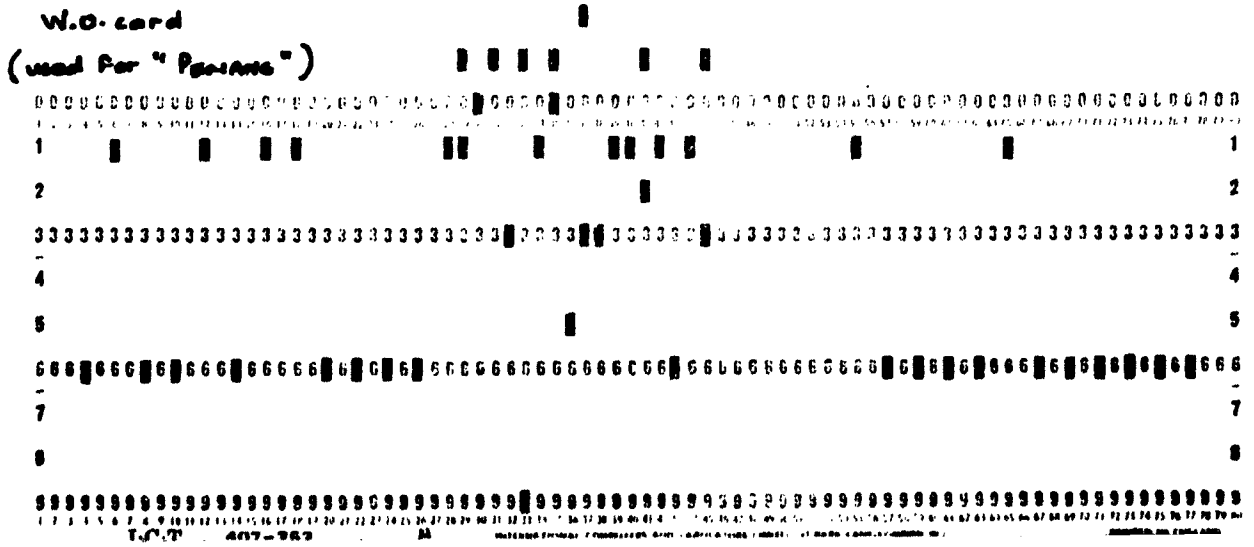
OF

W.O. card.

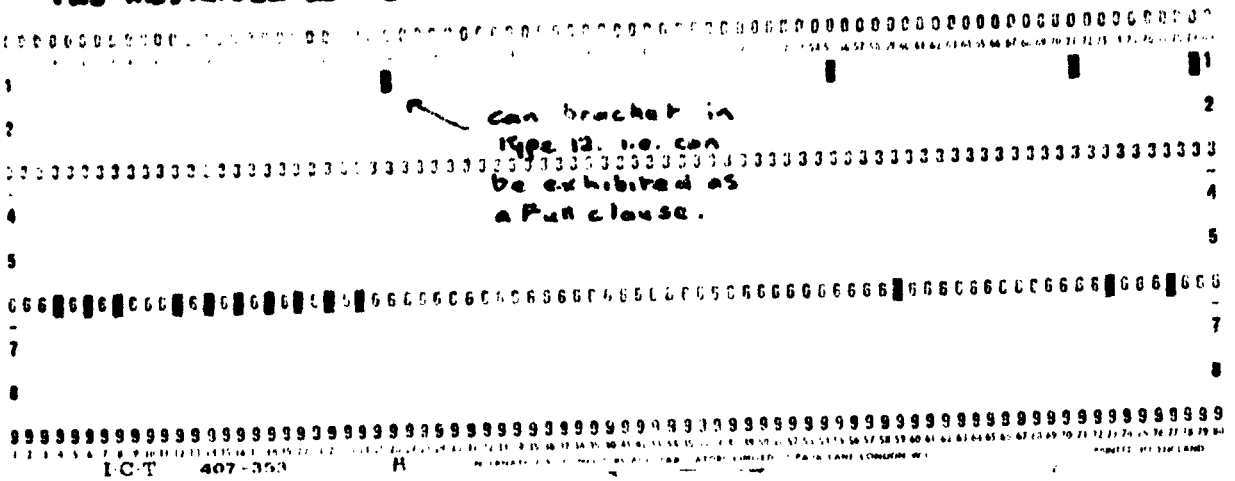


CAMBRIDGE

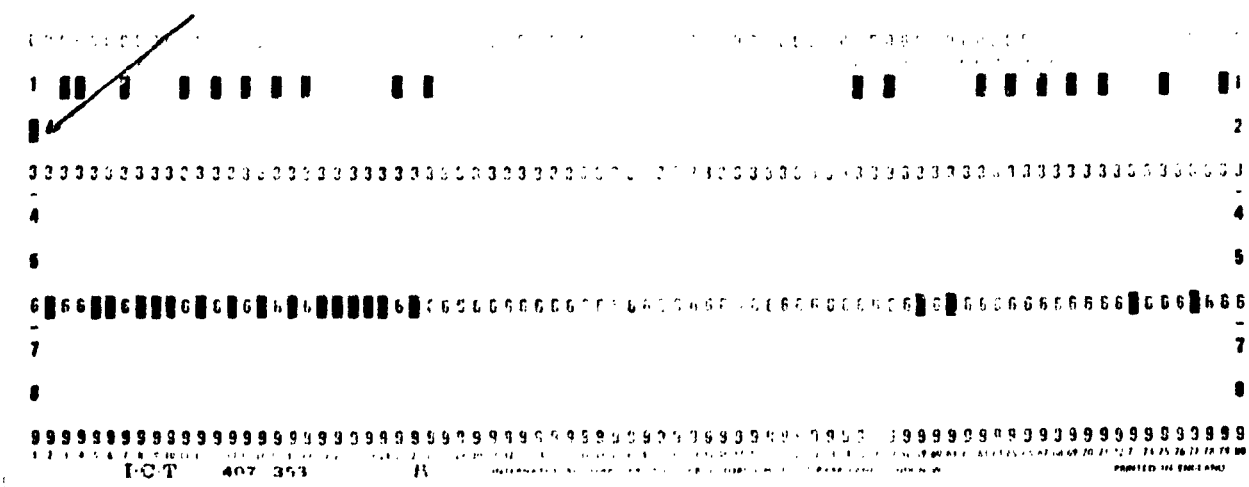
W.O. card
(used for "Pennine")



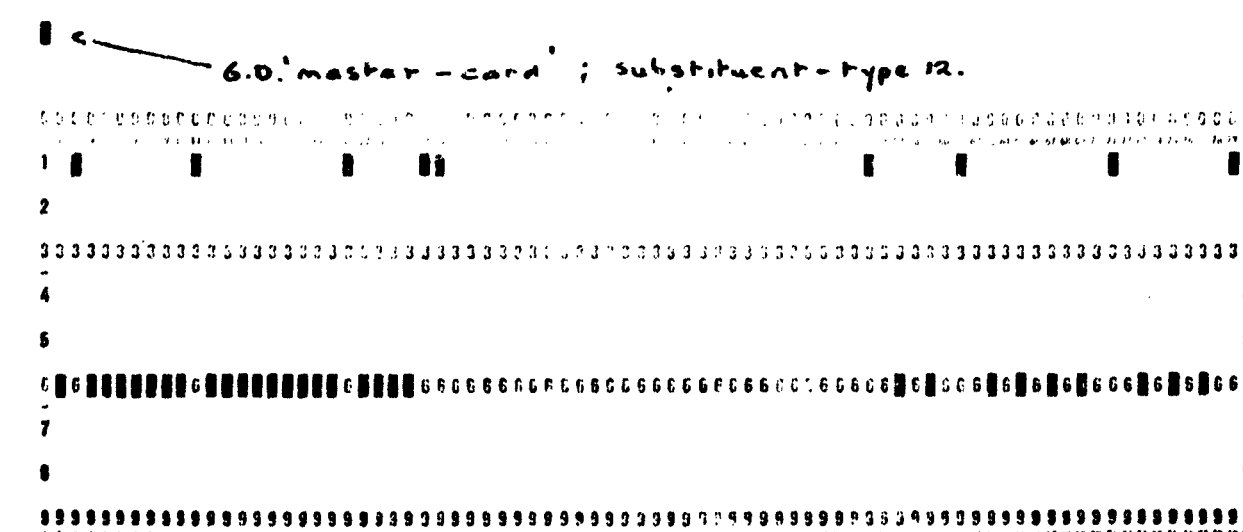
G.D. output card for the combination of the two 'master-cards' below.



G.D.'master-card'; substituent-type 2.



G.D.'master-card'; substituent-type 12.



3. The words of the sentence are written down the l.h.s. of the page in normal spoken order. The bracketting operations begin with the two lowest words in this order and work vertically up the sentence. The brackets are drawn to indicate that the words within them are being considered as a group. Their bracketting possibilities as indicated by the output card for the group of words concerned are shown by a series of numbers vertically in numerical order from the top. When on the l.h.s. of the bracket they indicate G-D possibilities, when on the r.h.s., W-O possibilities. Numbers not in parentheses indicate the types within which the group forms a valid bracket, those in parentheses show the types with which the group <u>could</u> form a valid bracket given certain properties of words yet to be considered.

Groups of words linked by a mark ⌶ have had a master-card substituted for them of the type indicated. This has been bracketted <u>as a substituent</u> within the bracket immediately to its right.

The symbol › indicates that it is <u>from this point upwards</u> that the bracketting is being done to the right of the mark as far as the next sign of this type.

⌶ Heavy signs, as left, indicate master-cards representing the bracketting together of master-cards themselves.

This is normally done when the beginning of the sentence has been reached, and we turn to consider how its longer substituents bracket together.

The "vanishing" of substituent numbers on the way up the sentence indicates the diminution of bracketting possibilities as more words are considered.

The sign 'X' indicates that an output card is blank i.e. that the bracket group is certainly invalid, in other words that the bracket is impossible, that is to say it is not a true group at all. After such a sign the next operation (substitution of a master-card, or re-bracketting from the last) sign) is recorded by a bracket drawn parallel to the last but to the right of it. The sign * at the head of a bracket indicates the application of iv) rule, (see P.88).

The sign § by a "master-bracket" indicates that this is, temporarily at least, stored as the bracketting of the sentence up to that point. The bracketting finally obtained by the method used is written at the bottom of each sentence-sheet. The sign † denotes application of the rule v) (see P.98).

4. ((A B)((C D) E))
12,11 7,6

The dictionary-checking consists in bracketting the sentence using the algorithms in the knowledge of the "correct" answer. This latter is provided by hand and displayed as above, the type of the brackets being written under the first parentheses. We take the intuitively "innermost" bracket and ensure that the G-D output card for its constituents "permits" a bracketting in type 6 (i.e. has a hole punched in row 1, column 14). We then immediately substitute MC(GD)6 and ensure that this can, with card (E) form a bracket in type 7. We then ensure that A and B together bracket in type 11, and finally ensure that MC(GD)11, MC(GD)7 together yield a card permitting bracketting in type 12 - i.e. that this internal bracketting is consistent with the whole standing as a sentence or "free clause". This sequence of operations is then repeated with the W-O cards.

If the result is the same as the one sought and obtained with the GD cards we assert that the intuitive bracketting of the sentence is at no point excluded by the information in the two syntactic dictionaries.

The sentences below, from Longmans-Green "Tales from

the East" p.26 are numbered in order from 1 - 10. The intuitive bracketting of each is reproduced at the head of the sheets bearing the bracketting results for that sentence.

---

(1) "In the northern part of Penang there is a little stone house near the sea.

(2) Near the little stone house there is a hut.

(3) The little stone house is there.

(4) The people call it Mat's hut.

(5) They tell the story of Ikan Dyong.

(6) He had two sons by his first wife.

(7) His second wife was young and very beautiful.

(8) She had blue eyes blue like the sea.

(9) She had a son Halim.

(10) His eyes were blue like the morning sky".

5. In general the bracketting procedure is applied according to the following rules:-

1. continue generating output cards until a blank card is obtained

2. when this happens examine the previous output card and substitute the master-card corresponding to the numerically lowest substituent type 'admitted' by that card.

3. If bracketting cannot continue with <u>this</u> card substitute the next 'permitted' master-card in numerical order.

4. If all this fails examine the previous output-card and repeat the above procedure.

5. If all this fails, even for the output-card for the first two substituents examined, re-bracket the sentence beginning with the last word but one of the string of words then under consideration: (repeat if necessary).

6. When doing this retain the permitted bracketting up to the point from which the bracketting begins.

7. On reaching the beginning of the sentence repeat the whole process with the substituents now obtained (the "second-level bracketting) to achieve a final permitted bracketting in type 12.

In strategy I, the implicit requirement of 1 - 7 above (i.e. that we should obtain the longest possible bracket) is modified using word order information in a restricted manner to reduce the amount of "breaking-back" of 5 above. Unless at least this is done the sentence will frequently not bracket at all.

Conjunctions are treated as follows: mark on both sides of the conjunction (determined by a given list of conjunctions) the smallest self-contained bracket. Endeavour to bracket both parts separately in the form of master-cards (unless the substituent is a single word) with the conjunction in type C. If this can be done we have a C bracket. The output-card for this group for further use is bracketting

the whole sentence containing the conjunction is formed from the logical sum of the master-cards for the two outer parts of the group:

i.e. "(young) and (very beautiful)"

Output-card is the union of the card for "young" and MC(GD)7 (for "very beautiful") for GD work and similarly for WO.

This is a weaker routine than taking the intersection or logical product, which may be more fruitful in future work, because for instance - "cats and books" is not a verb.

---

6. a. Notes on Strategy I.

The rules and "program", outlined here represent an attempt to avoid both long-winded 'back-trackings', and the problems of choice and store involved in using WO and GD information in parallel. It is true that the "stronger" the information used the less the chance of a 'bad choice' being made. But what is presented here uses a weaker 'binary' form of the WO algorithm (see infra) in the hope that a sacrifice of strength may pay in terms of simplicity i.e. by avoiding problems or using information in parallel while at the same time looking for the longest possible bracket. The latter

strategy would be all too simple without the last requirement - i.e. if we merely "made the appropriate bracket" after every word. This would, however, lead not only to redundancy but, in the case of sentences whose "deepest point" was not at the end, it would miss brackets altogether.

The efforts here add up to attempts to mark boundaries before beginning the bracketting proper which are observed when going 'up or down' the sentence, and is thus an attempt to preserve information in two different ways.

We require a) a success routine (i.e. we reach the beginning of the sentence) b) two limited failure routines i) does not bracket till after "back-tracking", ii) does not bracket even after "back-tracking". c) a routine for bracketting master substituent together after accepting certain masters and rejecting others, c) will be assumed to be of the form 'a + b', since substituents are not distinguished by their length. We assume further that the routines a, b, provide only one unamiguous bracketting of the sentence, after whose final bracketting by c the procedure stops.

The strategy differs from the general rules given in 5 in the following ways:

i) a master is substituted whenever a "break" (see infra) is reached

ii) this substitution is never 'unmade' when subsequently back-tracking i.e. when "moving the end" of the sentence it is never moved to a point within the substituent, but to the last "break" passed

iii)by recording the bracketting up to that point as a substituent to be bracketted later (on reaching the end of the sentence) as part of a second-level bracketting.

iv) a rule * that whenever an output card has governor holes but only in one or more of types 10, 11, 12 we move the end of the sentence > to above the last substituent considered and continue bracketting from there, while storing the output so far.

v) we include a rule that if the 'front' of the sentence is reached in such a way as to exhibit the <u>whole sentence</u> in some substituent type other than 12 we reject that type (and call the output zero).

This last rule enables us to deal with complex subjects and phrases which precede the subject of the sentence without tedious amounts of 'back-tracking'. In some sentences the rule has not been applied in order to compare the labour

involved in not using it. (See sentences 3, 8 and 10). The orders applying it in the 'program' can be deleted by deleting orders 40 and 41 and substituting '10' for '40' in order 22.

The 'breaks' are indicated in the results by horizontal red lines. They are of four types, conjunction, the beginning of the sentence, (both discussed above), binary word order and prepositions (see infra).

Binary word order - We precede the program with the application of the word order algorithm to all contiguous pairs of words in the sentence and mark a break between the members of those pairs the algorithm rejects entirely (i.e. where a 'final' word precedes an 'initial' word in every substituent type in which both occur). This gives a necessary and sufficient condition that a bracket be drawn at this point i.e. no connection rejected in this way can be reasserted by an output from a larger bracket.

Prepositions are marked by a break line preceding them in sentence order in a manner analogous to the conjunction rule. We assume an extensional list of propositional words and mark them so only if they are both preceding and followed by either a noun, an adjective or no word at all. It will be seen that strategy II. 'gets out' "preposition brackets" (type 2) more easily than this.

6. b. Notation

Consider a sentence of N words or basic substituents (i.e. initial units with listed dictionary entries) numbered 1 - N from the beginning of a sentence. Word-order (WO) breaks occur after words numbered a, b, c, (and N.). The strategy to be outlined provides acceptable words - output cards of a form $M_g^q(h)$ each having a stored G-D entry, where

q denotes the number of this unit numbered from the back of the sentence of accepted master units.

g denotes the position number of the first word in the substituent or unit ($1 \leq g \leq N$)

h denotes the current value of n (see below).

At any moment we deal only with two word-cards from the first n words of N after the word whose original place number is the current value of Θ (initially 1)

This enables us, after 'reaching' the beginning of the sentence from any point 'n' to return, set $\Theta = n$, N=n (i.e. the new 'n') and bracket up to 'n' from N.

The two cards in hand at any time are the word card $\underline{r}$ (numbered from the beginning of the n - word 'sentence') and the card X_r - (representing either a word or a substituent) representing the substituent last obtained or dealt with, this latter is therefore the $r+1^{th}$ effective substituent.

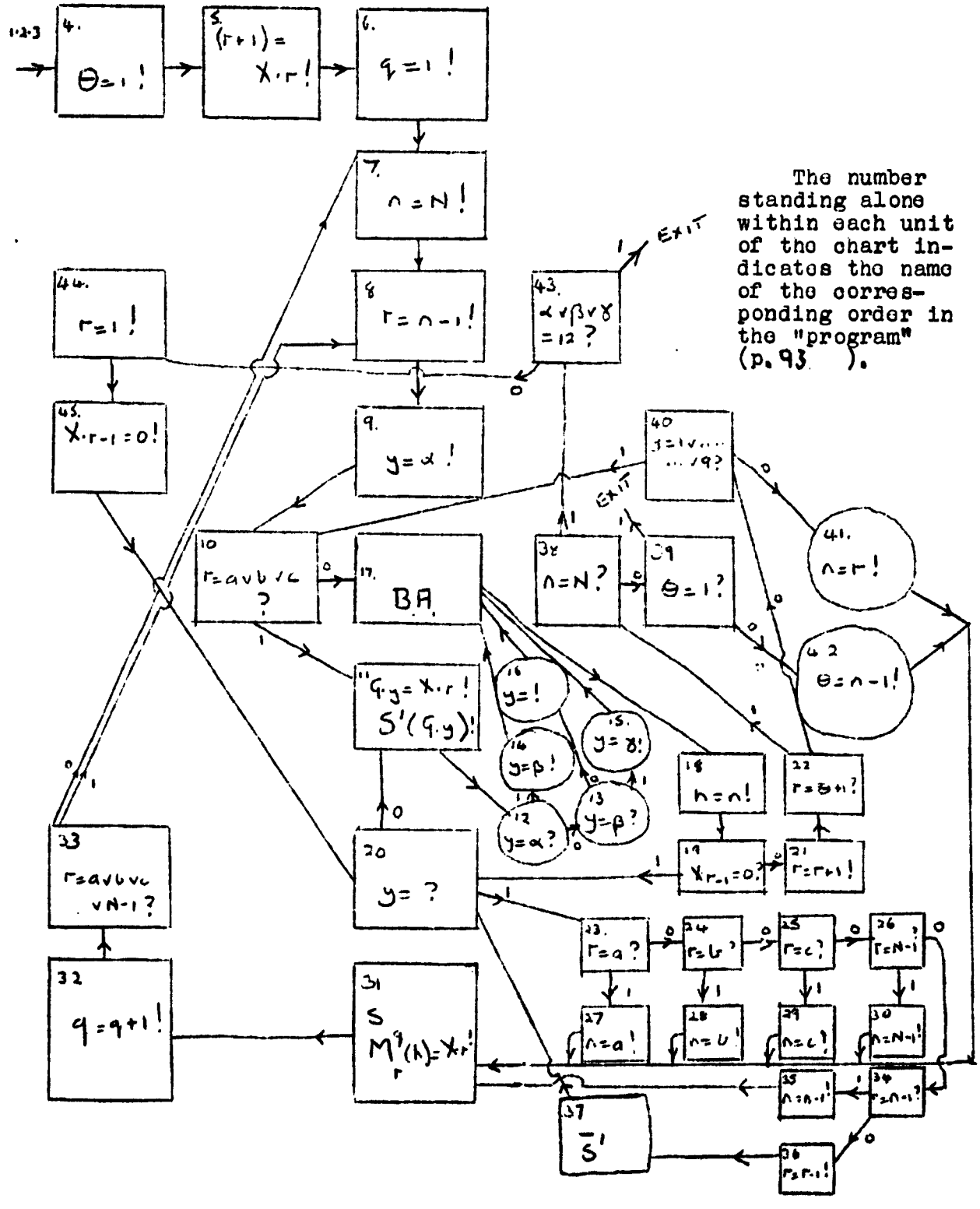
The GD bracketting algorithm (BA) is performed on these two cards to produce the output card X_{r-1}, (which is in turn compared with the $r-1^{th}$ word card). (r) and (X) will refer to the cards in question. Questions like $(X_k) = ?$ will ask whether there are any governor or dependent holes on (X_k). The value of the variable y at any time refers to the position number ($1 \leq y \leq 12$) of the furthest left of the 'unused' (i.e. not so far considered by the process) governor holes on X_r - the set of them being labelled α, β, γ etc. ($\alpha < \beta < \gamma$). The symbol 'G'_y will be taken to indicate the master-card for the type y.

S denotes a 'store-operation'.

By appropriate re-assignment of variables (not given) when n = 0, the "second-order bracketting" of the 'M' units is achieved by the same series of questions and orders.

An exclamation mark (!) indicates an order which is executed and the next order executed is the one whose number occurs in the bracket following the exclamation mark. A question mark (?) indicates a question. If the answer is 'yes' the procedure moves to the order whose number occurs first in the bracket following the question mark. Conversely if the answer is 'no' the procedure moves to the order indicated by the second digit in the bracket.

The notation is a simplified form of the Parker-Rhodes Algorithm Notation.

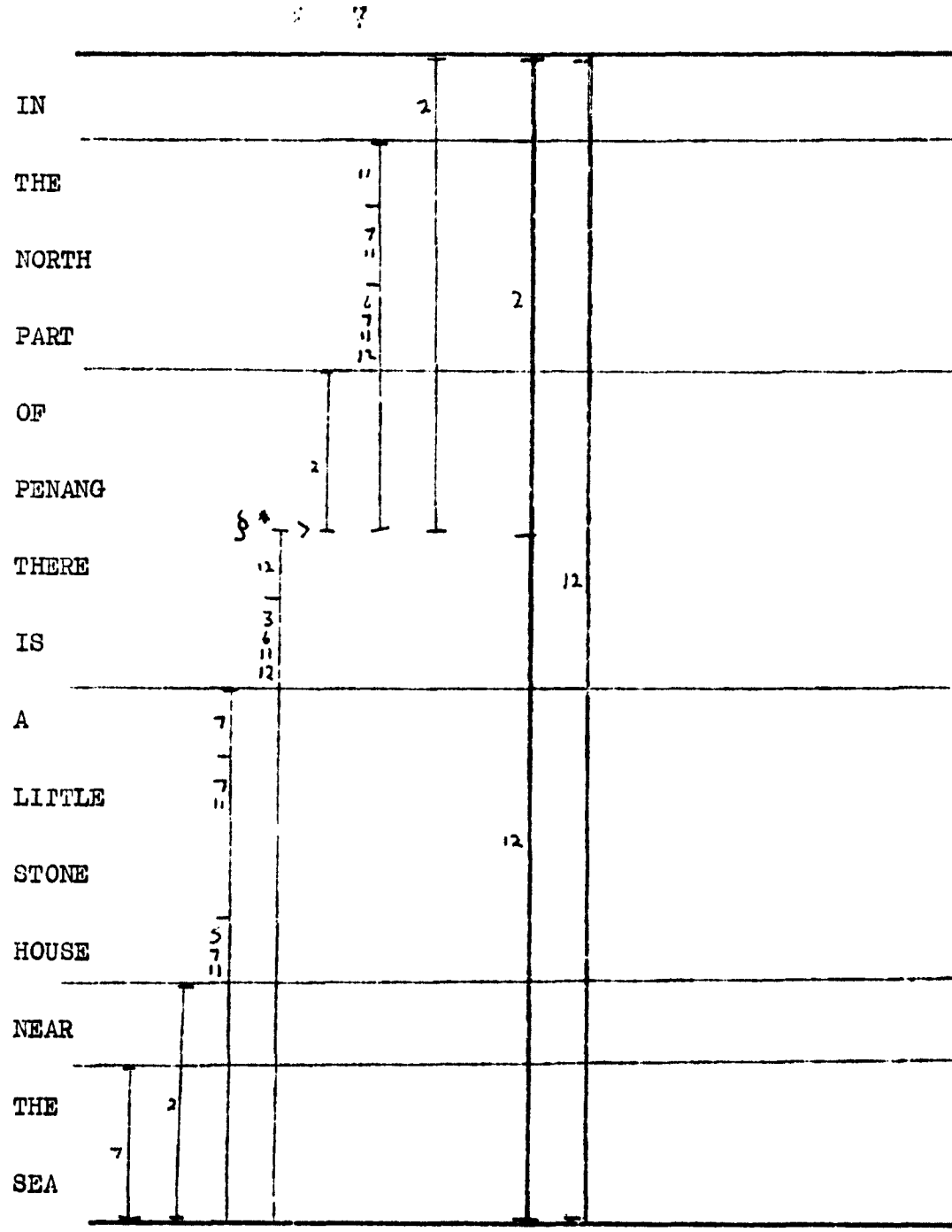


The number standing alone within each unit of the chart indicates the name of the corresponding order in the "program" (p. 93).

6. d. Program

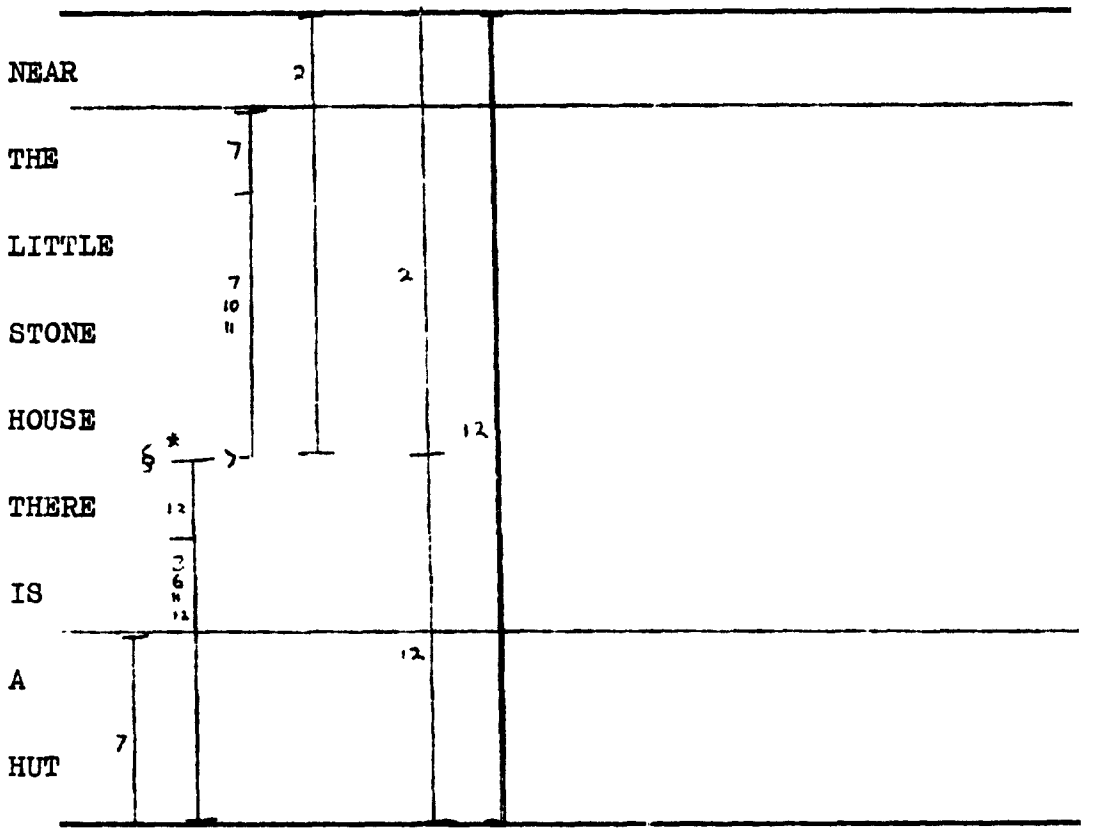
1. =(r;0) 0 ⟨ r ≤ n ≤ N
2. = /X/
3. = /M/
4. ϕ = !
5. (r+1) =(X.r) !
6. q =1 !
7. n=N!
8. r=n-1 !
9. y=α !
10. r= a^b^c ? (11,17)
11. (G.y)=(X.r) ! S' (G.y)
12. y=α ? (14,13)
13. y=β ? (15,16)
14. y=β ! (17)
15. y=γ ! (17)
16. y=' (17)
17. BA ((r),(X.r)) !
18. h=n !
19. (X.r-1)=? (20,21)
20. y=? (23,11)
21. r=r+1 !
22. r= ϕ+1 ? (38,40)
23. r=a ? (27,24)
24. r=b ? (28,25)
25. r=c ? (29,26)
26. r=N-1 ? (30,34)
27. n=a ! (31)
28. n=b ! (31)
29. n=c ! (31)
30. n=N-1 ! (31)
31. S $M_r^q(h)$ =X.r !
32. q=q+1 !
33. r=a^b^c^N-1 ? (8,7)
34. r=n-1 ? (35,36)
35. n=n-1 ! (31)
36. r=r-1 !
37. unstore S' (31)
38. n=N ? (43,39)
39. ϕ = 1 ?(EXIT, 42)
40. y=1^2^3^4^5^6^7^8^9 ? (10,41)
41. n=r ! (31)
42. ϕ =n-1 ! (31)
43. α^β^δ =12↑ (EXIT,44)
44. r=1 !
45. (X.r-1) = ! (20)

Results . 1. 1227 2 12 11 7
((IN(THE NORTH PART(OF PENANG))))(THERE(IS(A
LITTLE STONE HOUSE) (NEAR (THE SEA)))))



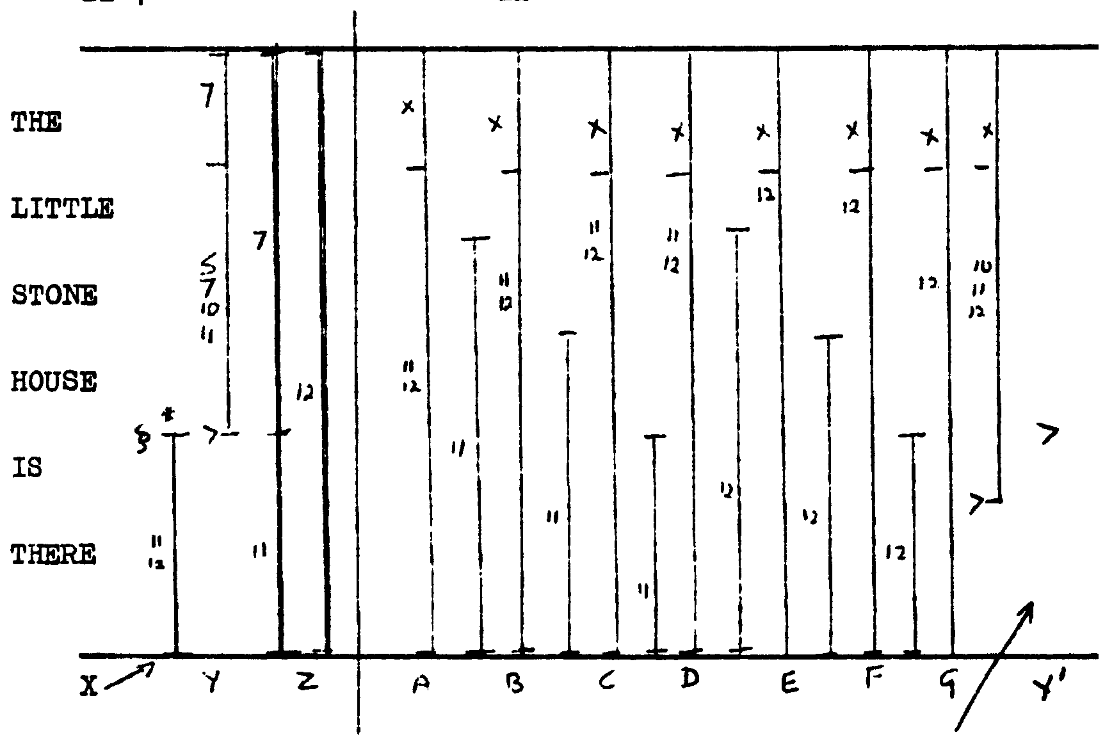
((In (the north part(of Penang)))(there is(a little stone house(
2 7 2 12 7 2
near(the sea)))))
7

2. ((NEAR(THE LITTLE STONE HOUSE))(THERE(IS(A HUT))))
12 2 7 12 11 7



((Near(the little stone house))(there is(a hut)))
12 2 7 12 7

3. ((THE LITTLE STONE HOUSE)(IS THERE))
12 7 11



(i)

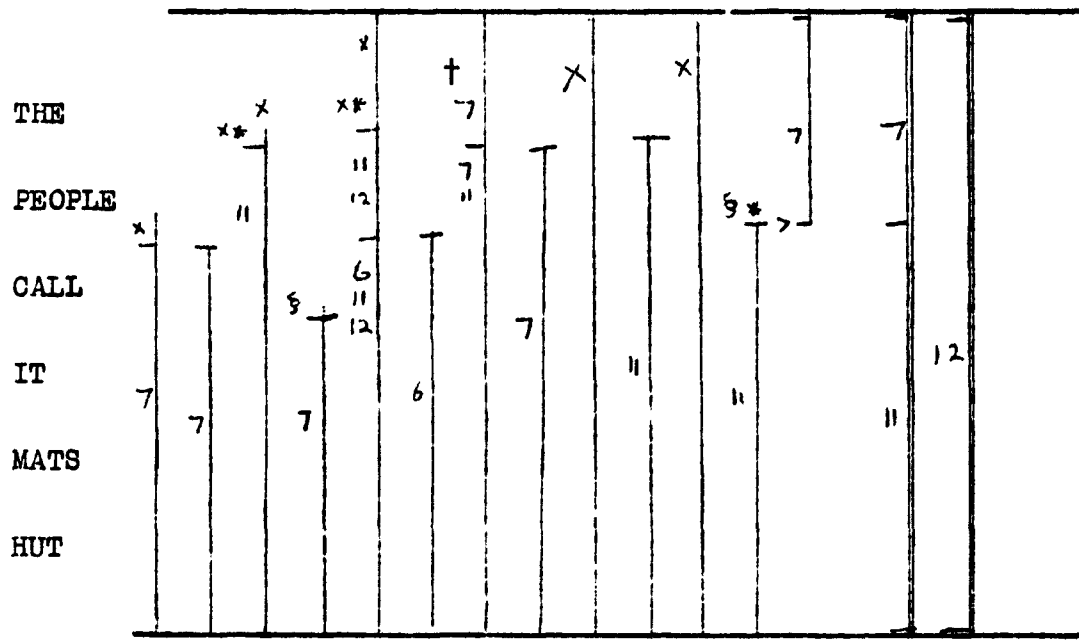
(ii) Repeat cycle
A - G
re-enter at
Y' (corr. to Y)

Brackotting displayed after π line shows 'back-breaking' procedure had we not applied * rule at X.

i) ((The little stone house)(is there))
12 ← 7 11

ii) as for i)

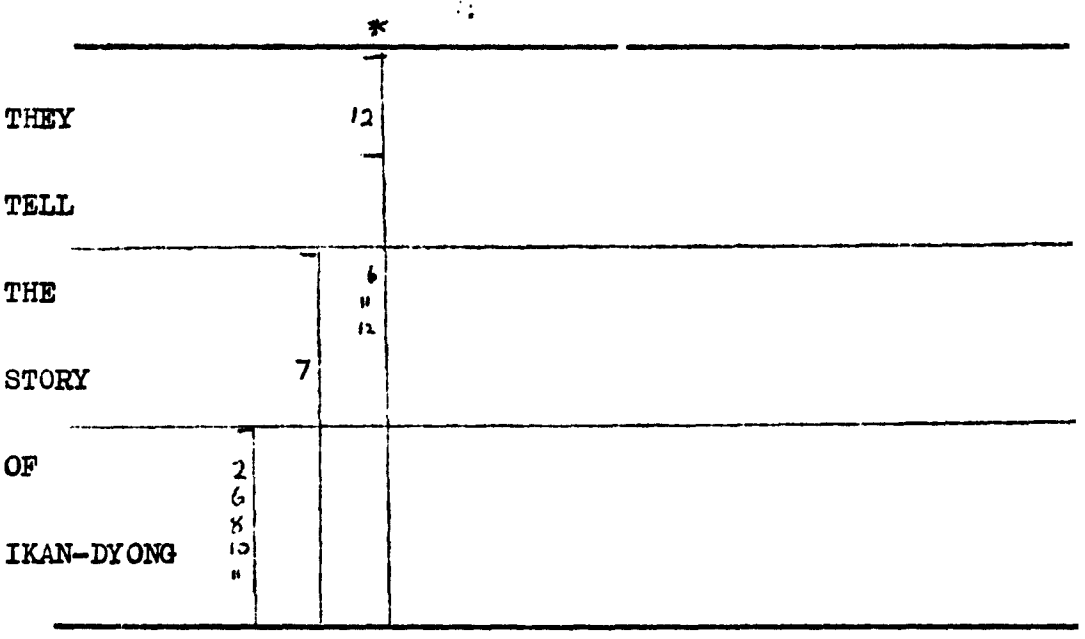
4. ((THE PEOPLE)(CALL IT (MATS HUT)))
12 7 11 7



x = indicates that rule * was tried at this point and failed.

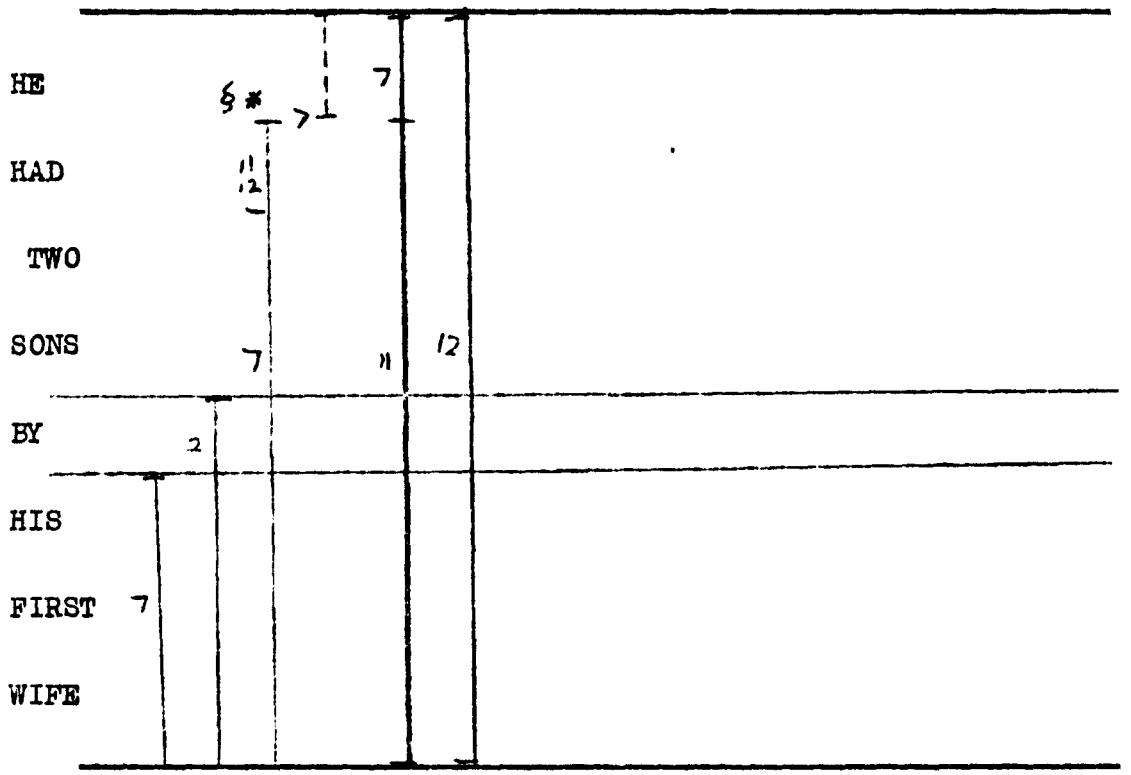
((The people)(call(it Mats hut))).
12 7 11 7

5. (THEY(TELL(THE STORY(OF IKAN - DYONG))))
12 11 7 2



(They tell(the story(of Ikan-Dyong)))
12 7 2

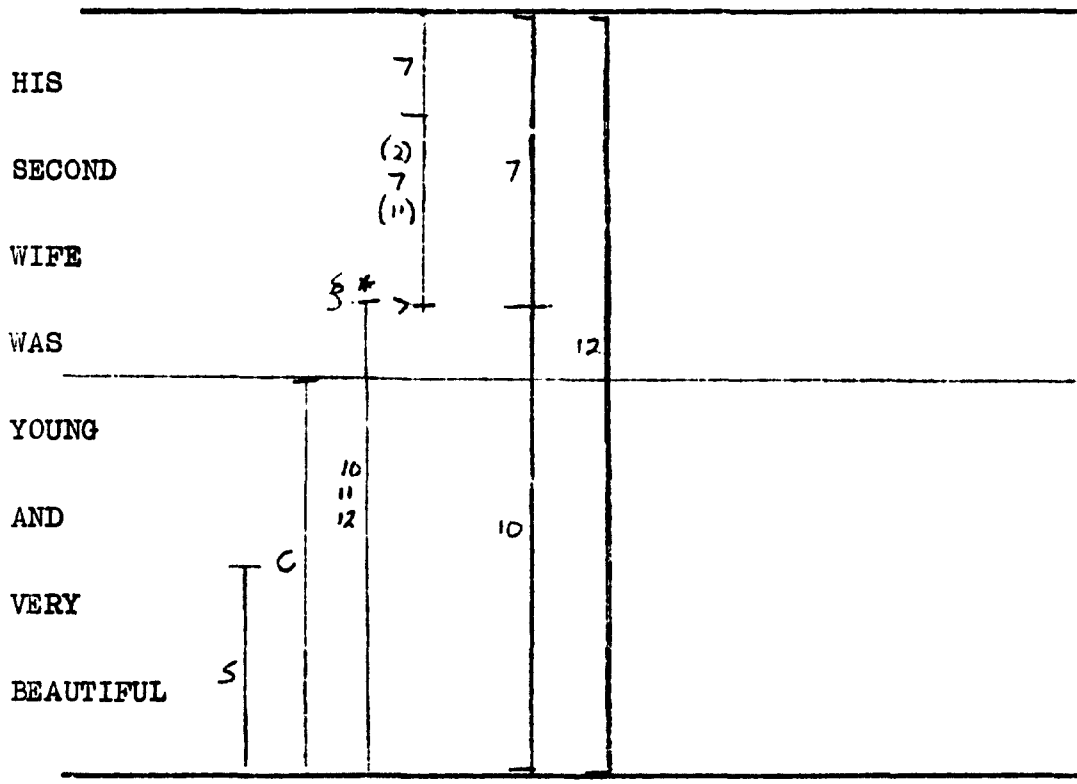
6. (HE(HAD(TWO SONS(BY(HIS FIRST WIFE)))))
12 11 7 2 7



(He(had two sons(by(his first wife))))
12 11 2 7

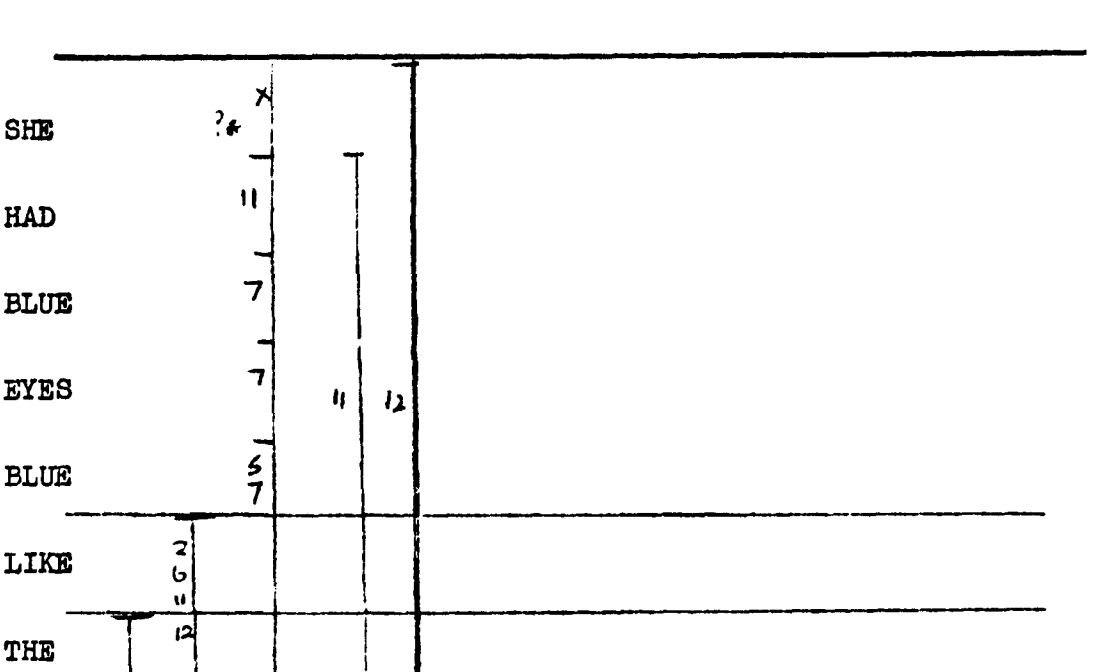
Application of the rule * is trivial here, the procedure would have worked without it on substitution of an 'll-master'.

7. ((HIS SECOND WIFE)(WAS(YOUNG AND(VERY BEAUTIFUL))))
12 7 10 C 5



((His second wife)(was(young and(very beautiful))))
12 7 10 C 5

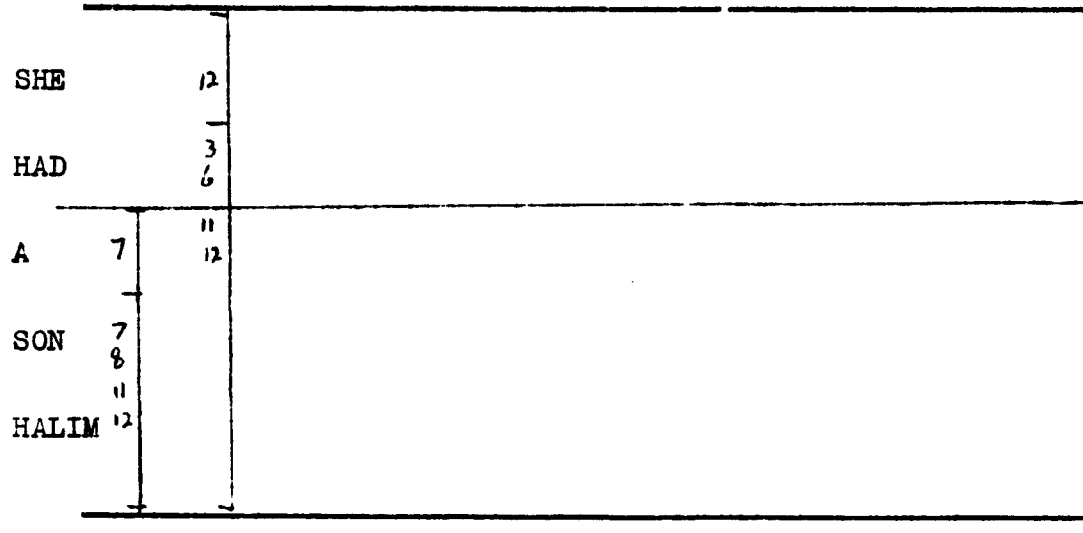
8. (SHE (HAD(BLUE EYES)(BLUE(LIKE(THE SEA)))))
1 2 11 5 2 7



(She(had blue eyes blue(like(the sea))))
12 11 2 7

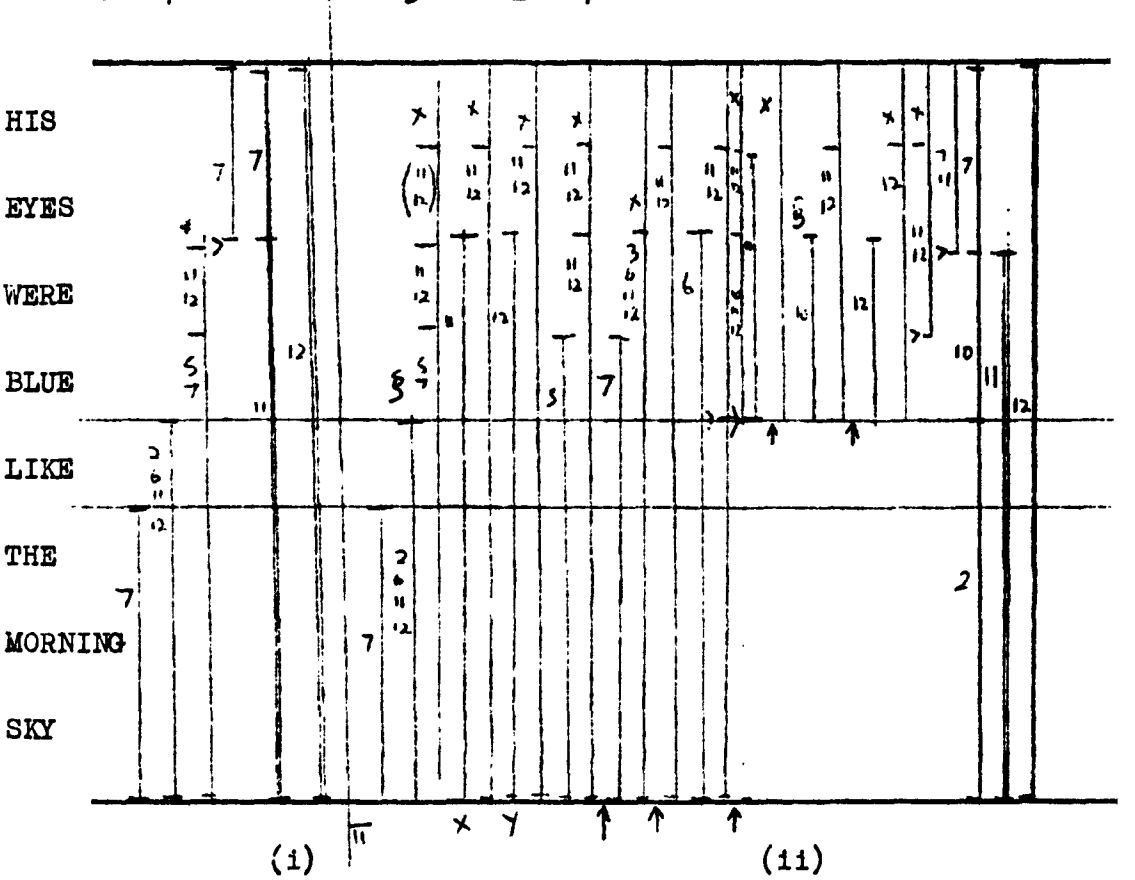
? * rule could have been applied trivially here, as in sentence 6. - it makes no difference.

9. (SHE(HAD(A SON HALIM)))
12 11 7



(Sho had (a son Halim))
12 7

10. ((HIS EYES)(WERE(BLUE(LIKE(THE MORNING SKY)))))
12 7 11 5 2 7



(i) ((His eyes)(were blue(like(the morning sky))))
12 7 11 2 7

(ii) ((His eyes)((were blue) (like (the morning sky))))
12 7 11 10 2 7

The arrows (↑) indicate a repeat (not shown) of the cycle X-Y at that point.

8. Notes on Strategy II.

1. Output cards are generated at the same time for each pair of substituents considered for both GD and WO information.

2. If either type of information yields a blank card a substitution of masters is made at that point for both types of information.

3. The master chosen is always that of the substituent type of lowest number occurring in the intersection of the sets of numbers of the substituent-types permitted by the two output cards generated at that point. The sets are of the "distinct possibilities of association at this point" only i.e. not those that require us to look at words not so far considered.

4. The WO results at each stage are written on the right of the "bracketting line", the GD on the left.

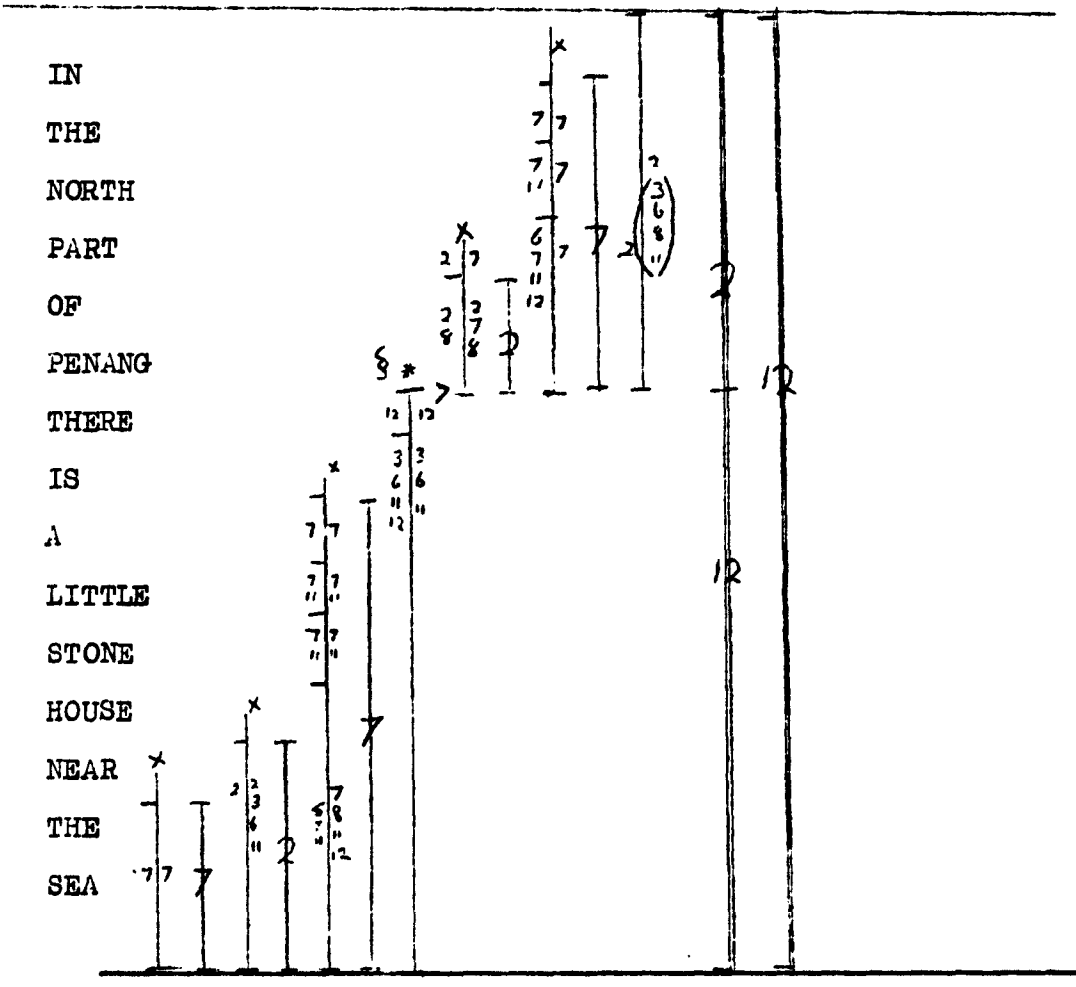
5. If the intersection of total possibilities at any stage (i.e. not the one specified in (3)) is zero, we treat it as we would the appearance of a blank card for either type of information.

6. The rule * is applied to avoid back-tracking (as in I.) in this form; - - - if the intersection (specified in 3) consists only of one or more of the types 10, 11, 12 we continue bracketting the sentence starting from above the last word substituent considered. At the same time we store the bracketting of the sentence up to this point so far obtained.

7. The conjunction rule is applied as in I., and there is no preposition rule. Other notation is as in I.

9 Results II.

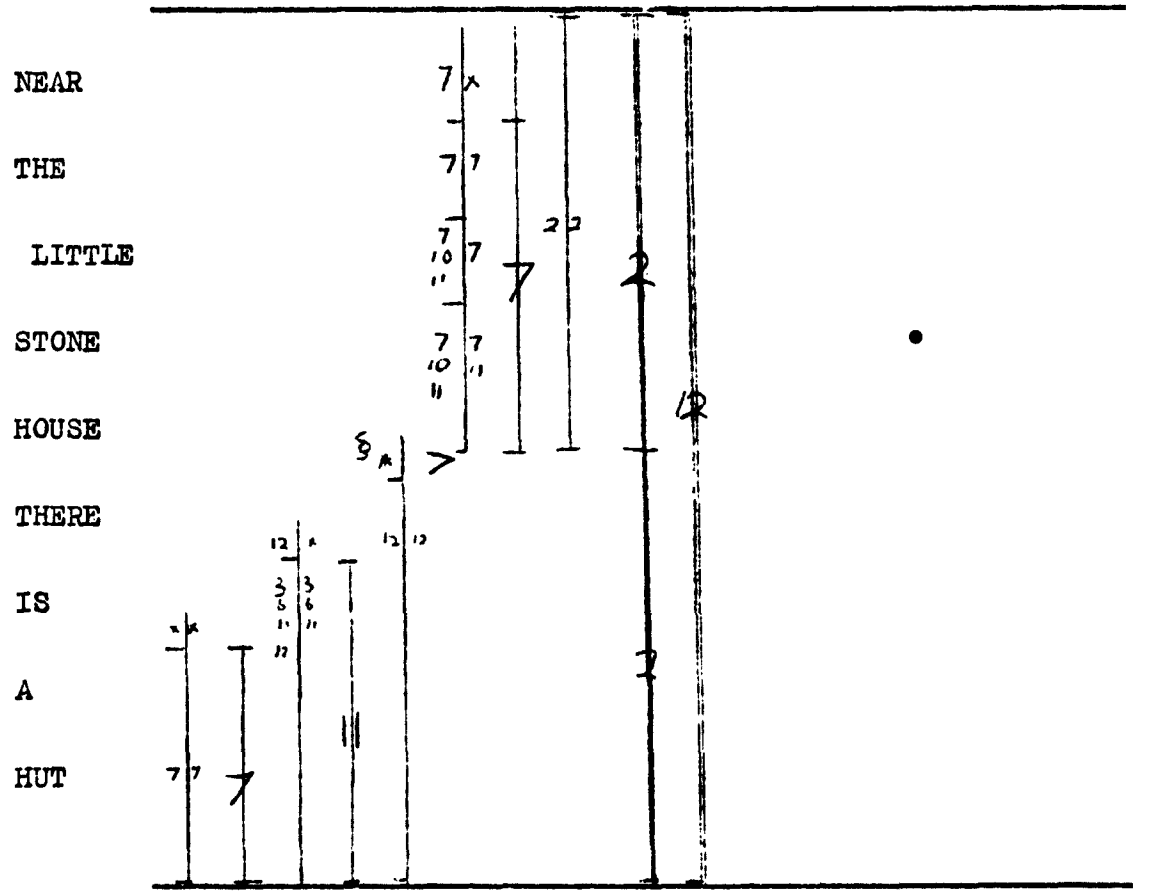
1. 12 2 7 2 12 11 7
((IN(THE NORTH PART(OF PENANG)))(THERE(IS(A LITTLE STONE HOUSE) (NEAR (THE SEA)))))
2 7



((In (the north part (of Penang) (there is (a little stone
12 2 7 2 12 7

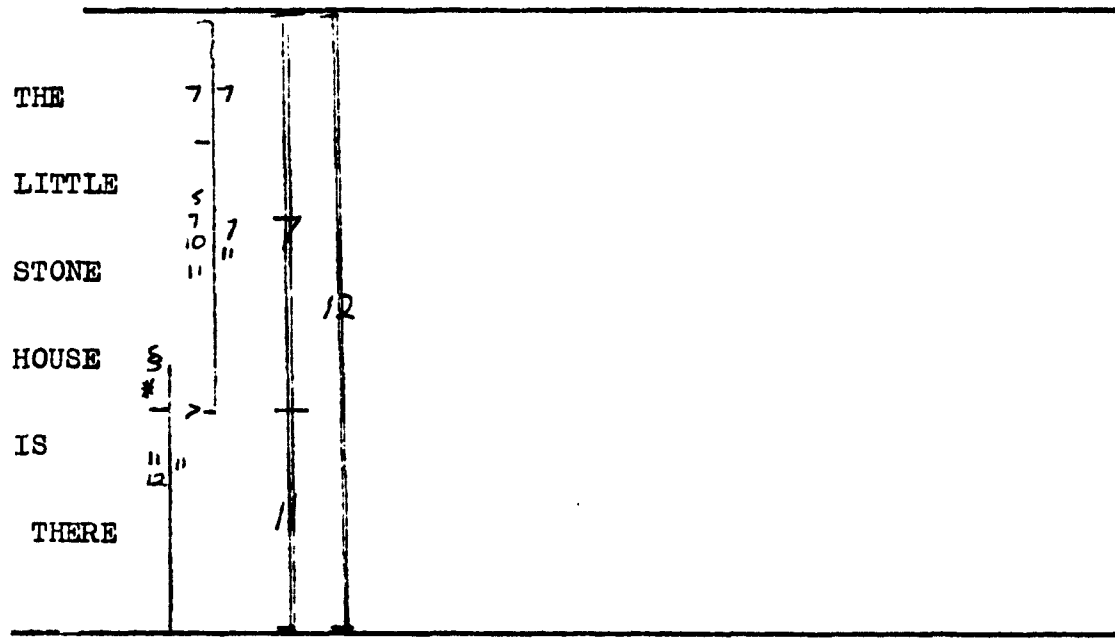
house (near (the sea)))
2 7

2. ((NEAR(THE LITTLE STONE HOUSE))(THERE(IS(A HUT))))
12 2 7 12 11 7



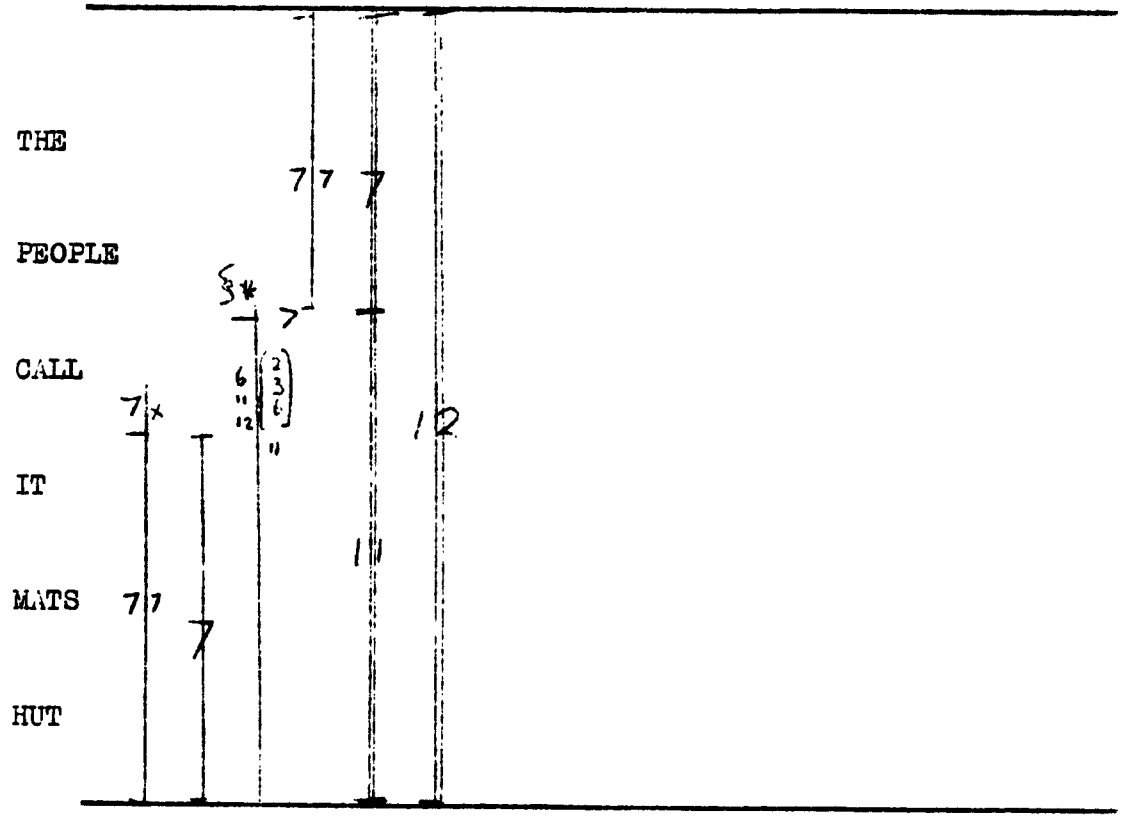
((Near (the little stone house)) (there(is(a hut))))
12 2 7 12 11 7

3.. ((THE LITTLE STONE HOUSE) (IS THERE))
12 7 11



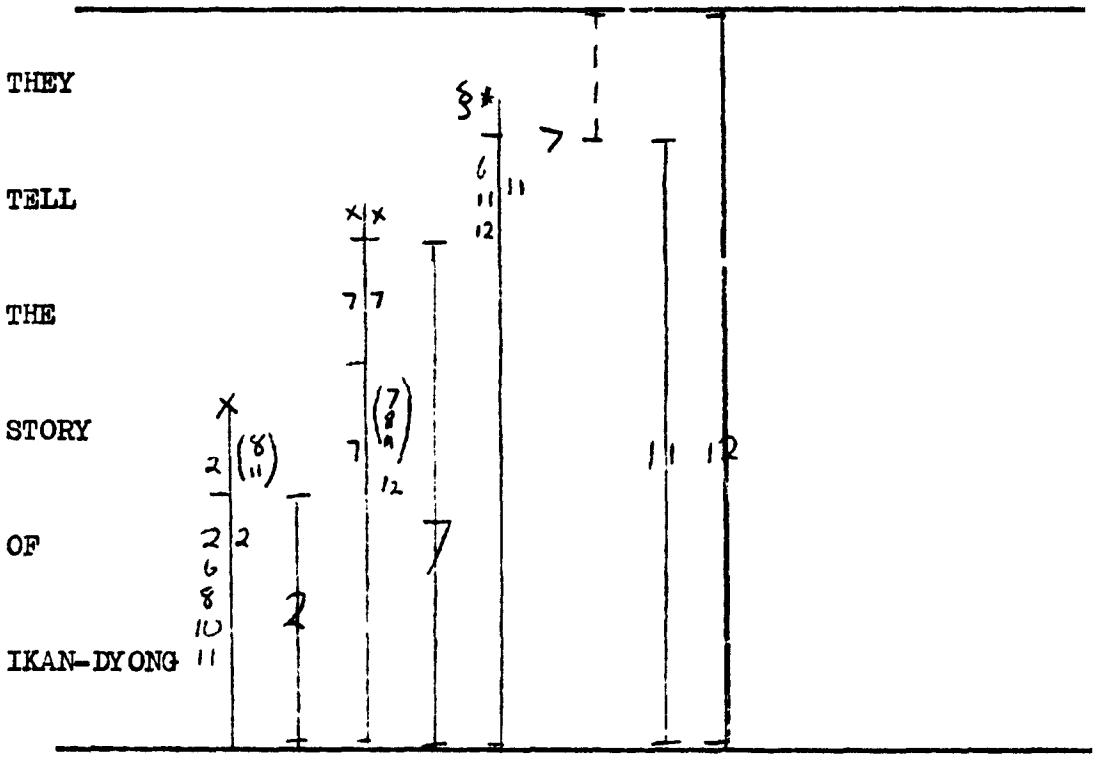
((The little stone house)(is there))
12 7 11

4. ((THE PEOPLE) (CALL IT (MATS HUT)))
12 7 11 7



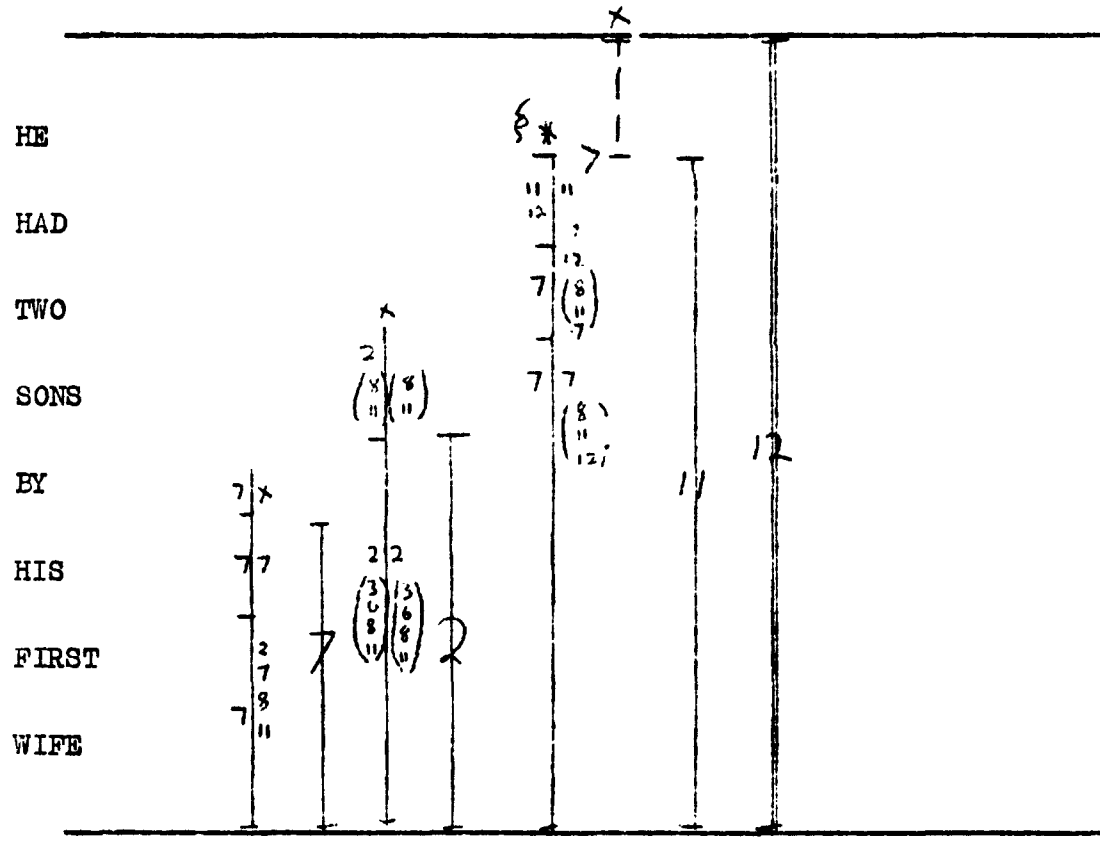
((The people) (call (it Mat's hut)))
12 7 11 7

5. (THEY(TELL(THE STORY(OF IKAN-DYONG))))
12 11 7 2



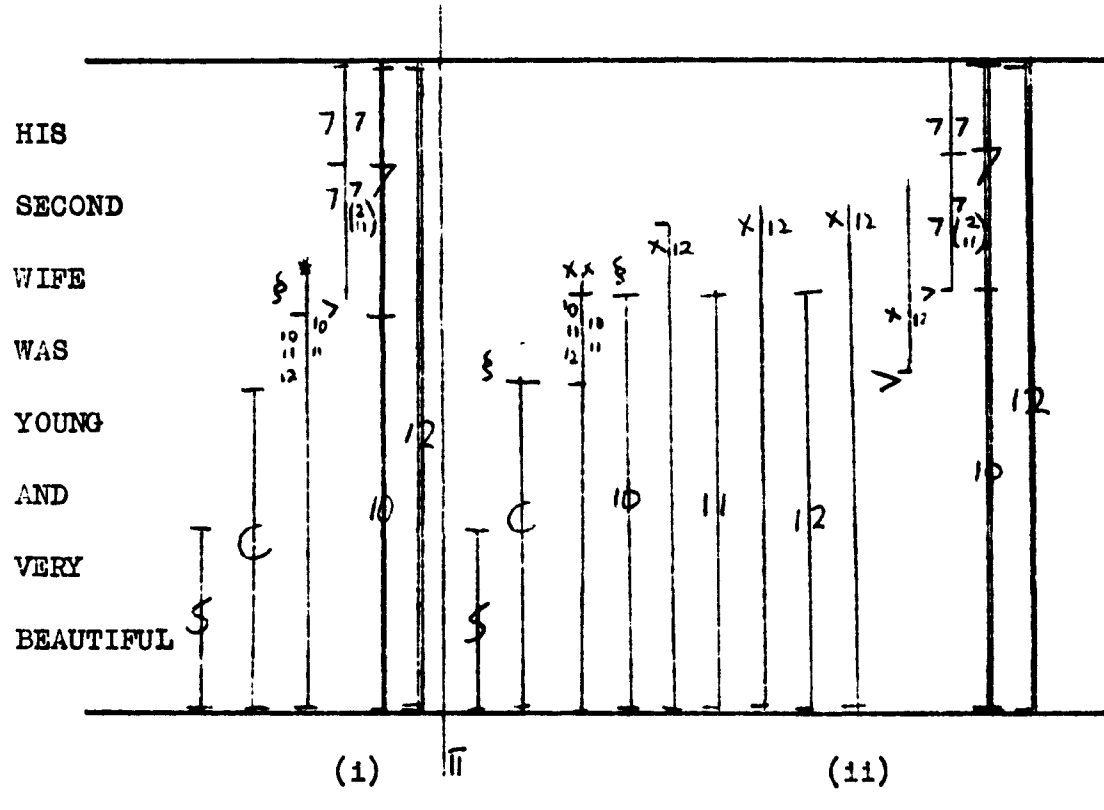
(They(toll(the story (of Ikan-Dyong))))
12 11 7 2

6. (HE(HAD(TWO SONS(BY(HIS FIRST WIFE)))))
12 11 7 2 7



(He(had two sons(by(his first wife))))
12 11 2 7

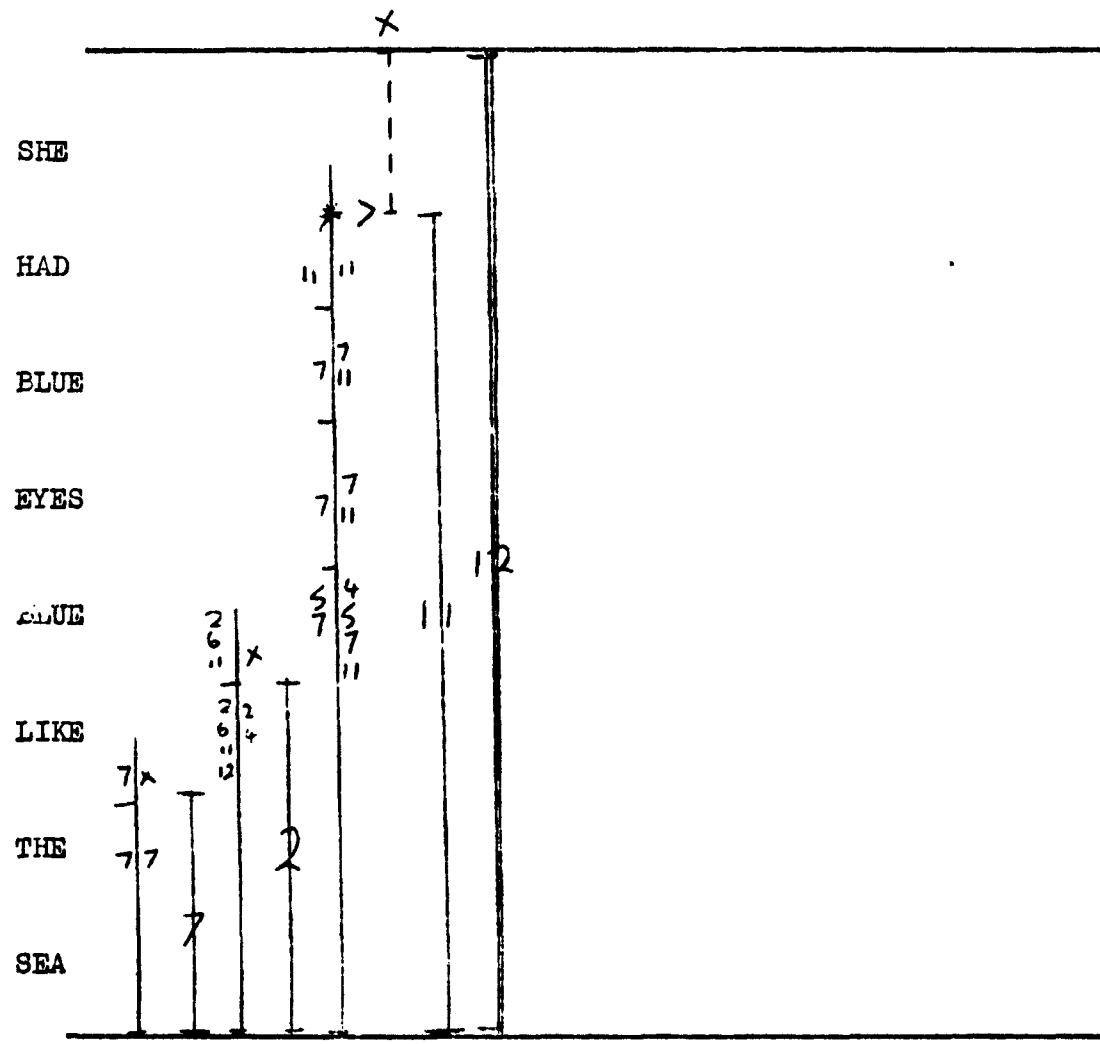
7. ((HIS SECOND WIFE) (WAS(YOUNG AND(VERY BEAUTIFUL))))
12 7 10 C 5



(ii) examines the bracketting not using the 10-11-12 rule. (It assumes that the 'C' bracket constitutes a unit which cannot be 'unmade').

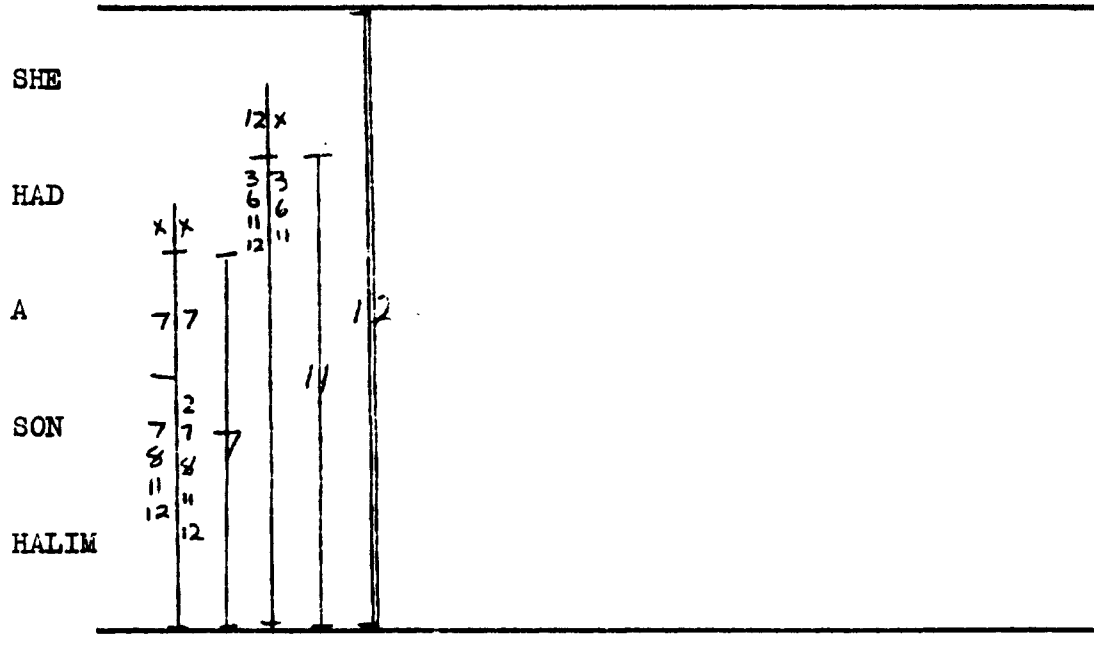
(i)
&
(ii) ((His second wife)(was(young and(very beautiful))))
12 7 10 C 3

8. (SHE (HAD(BLUE EYES) (BLUE(LIKE(THE SEA)))))
12 11 7 5 2 7



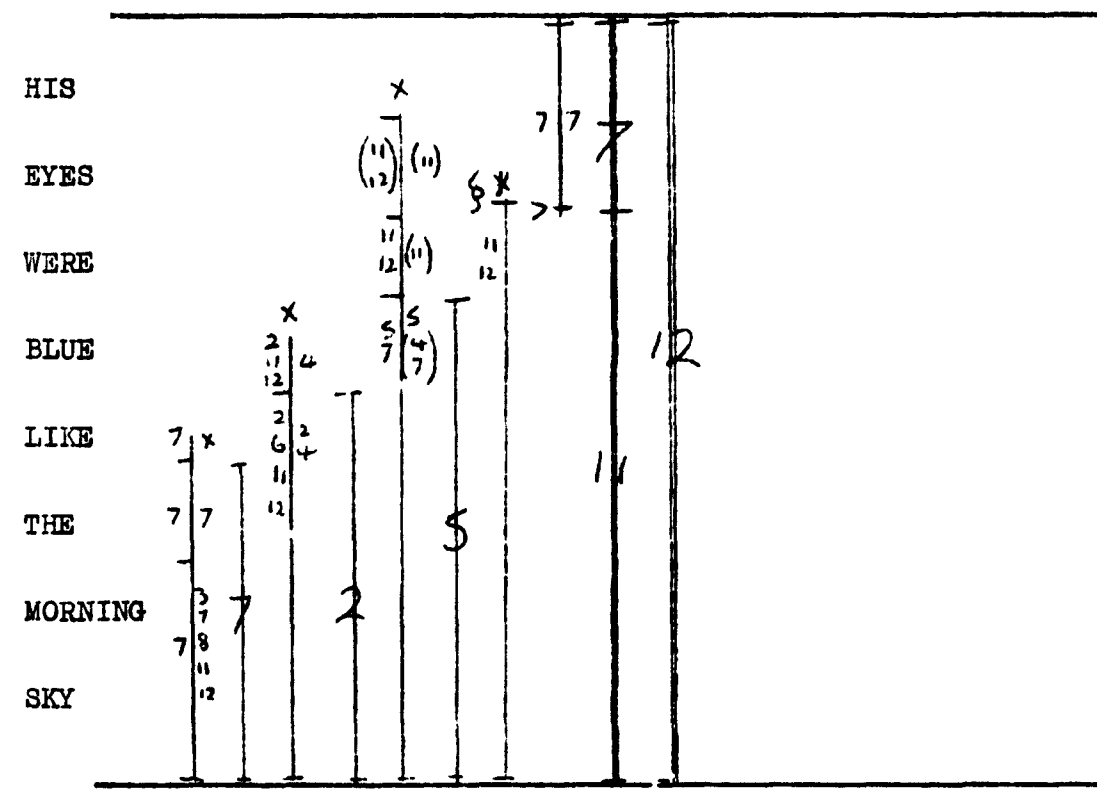
(She (had blue eyes blue (like (the sea))))
12 11 2 7

9. (SHE(HAD(A SON HALIM)))
12 11 7



(She (had(a son Halim)))
12 11 7

10. ((HIS EYES)(WERE(BLUE(LIKE (THE MORNING SKY)))))
12 7 11 5 2 7



((His eyes)(were(blue(like(the morning sky)))))
12 7 11 5 2 7

10 CONCLUSION

It will be seen by comparing the proportions of "success brackets" obtained by strategies I and II that II is better in two ways:

1) it requires pre-editing only for conjunctions,

2) the results displayed in preceding pages show that II "misses" five brackets (of which one is a plausible alternative bracketting), while I "misses nine (of which two are plausible alternative brackettings).

I think it has been demonstrated here that the decrease in back-tracking obtained amply justifies the use of some form of the "10-11-12" rule. (See p. 88 (iv)).

BIBLIOGRAPHY

1. George Boole : The Mathematical Analysis of Logic. 1847.

2. George Boole : An Investigation of the Laws of Thought. 1854.

3. R.G. Collingwood : An Autobiography. 1939.

4. Alonzo Church : 'An unsolvable problem of Elementary number theory' in Amer.J.of Maths. 1936.

5. D. Hilbert and P. Bernays : Die Grundlagen der Mathematik. Vol. 1. 1934.

6. D. Hilbert and W. Ackermann : Grundzüge der theoretischen Logik. 1938.

7. C.I. Lewis : A Survey of Symbolic Logic. 1918.

8. J. Lukasiewicz : 'Philosophische Bemerkungen zu mehrwertigen Systemen des Aussagenkalkuls' in Comptes Rendus des Seances de la Soc. des Sci. et de Lettres de Varsovie. 1930.

9. A.F. Parker-Rhodes : 'Is there an interlingual Element in Syntax?' Proceedings of the Int. Cong. of
& Linguists. 1962. 'A New Model of Syntactic Description' in Proc. of the First Inter. Conf. of Machine Trans. of Lang. and Applied Lang. Analysis;
10 Teddington. 1961. (10)

11. C.S. Peirce : 'On the Algebra of Logic' in Amer. J. of Maths. 1880.

12. E.L. Post : 'A general Theory of Elementary Propositions' in Amer. J. of Maths. 1921.

13. H. Reichenbach : Philosophical Foundations of Quantum Mechanics. 1946.

14. C. Shannon : 'A Symbolic Analysis of Relay and Switching Circuits' in Trans. of the Amer. Inst. of Elect. Engin. 1938.

15. H.M. Sheffer : 'A Set of Five Independent Postulates for Boolean Algebras'in Trans.Amer.Math.Soc. 1913.

16. E. Schröder : Vorlesungen über die Algebra der Logik. 1890-1905.

17. A. Tarski : 'Der Wahrheitsbegriff in den Formalisierten Sprachen' in Studia Philosophica. 1935.

18. A.M. Turing : 'On Computable Numbers' in Proc. Lond. Math. Soc. 1936-1937.

19. C.F. von Weizsäcker : Komplementarität und Logik' in Die Naturwissenschaften. 1955.

20. L. Wittgenstein : Tractatus Logico Philosophicus. 1922.

21. V. Yngve : 'The Depth Hypothesis' in Procs. of the Symp. on Applied Maths. Vol. XII. 1961.

22. C.F. Hockett : A Course of Modern Linguistics. 1958.

APPENDIX I.

M.L. 159

CLRU SYNTAX DICTIONARY

for

E N G L I S H

Vocabulary No. 1

Revision of June 1962

Cambridge Language Research Unit
20 Millington Road
Cambridge, England

SUMMARY OF CODING

List of Substituent Types

1. Adverbial Group
2. Adverbial O-Clause (also functions as adjunctive)
3. Adverbial S-Clause (also functions as nominal)
4. Adverbial Subjunct

5. Adjunctive Group
6. Adjunctive S-Clause

7. Nominal Group
8. Nominal O-Clause (also has adverbial & adjunctive uses)
9. Nominal Subjunct

10. Operative Group
11. Operative S-Clause or Predicate
12. Full Clause

Definition of Coding Digits

Word-Order : 1 = can be initial, but may not follow a "2"
2 = can be final and may not precede a "1"
3 = can occur anywhere
0 = cannot occur at all

Conjunction	:	1	= can be joined by a conjunction
		2	= can act as a conjunction (narrow sense)
		3	= can both join and be joined
		0	= cannot be joined by a conjunction
Dependency	:	1	= can be dependent only
		2	= can be governor only
		3	= can be either
		0	= cannot occur at all

S Y N T A X D I C T I O N A R Y

Revised June 1962

Word	A.L.N.	Participation Class: Word-Order	Conj	Dependency
A		0000 00 100 000	1	0000 00 100 000
abstract		0200 10 300 230	1	0100 20 300 320
abyss		0000 00 300 000	1	0000 00 200 000
accomplished		0002 30 100 230	1	0001 20 100 320
accordance		0200 00 320 021	0	0100 00 210 011
activities		0200 00 320 021	1	0100 00 210 011
adopted		0002 30 100 230	1	0001 20 100 320
adored		0002 30 100 230	1	0001 20 100 320
age-old		0002 30 100 020	1	0001 20 100 010
ages		0220 02 300 333	1	0120 02 200 333
agriculture		0200 00 320 021	1	0100 00 210 011
all		0000 30 100 020	1	0000 20 100 010
allows		0000 00 000 210	1	0000 00 000 220
alphabets		0200 00 320 021	1	0100 00 210 011
also		1000 10 000 113	3	1000 10 000 111
amount		0220 02 200 312	1	0120 02 200 320
an		0000 00 100 000	1	0000 00 100 000
analysis		0200 00 320 021	1	0100 00 210 011
animal		0002 30 200 020	1	0001 20 300 010
another		0202 00 130 021	1	0101 00 110 011
anthoropology		0200 00 320 021	1	0100 00 210 011
any		0202 00 130 021	1	0101 00 110 011
apologize		0020 02 020 212	1	0020 02 020 322
approached		0022 32 100 232	1	0021 22 100 322
are		0220 02 320 333	1	0120 02 210 232
Aristotle		0200 00 320 021	1	0100 00 210 011
arm		0020 02 200 310	1	0020 02 200 320
articles		0200 00 320 021	1	0100 00 210 011
as		0101 10 000 000	0	0202 10 000 000
asked		0022 32 100 232	1	0021 22 100 322
attempts		0200 00 300 211	1	0100 00 200 333
authentic		0002 30 100 020	1	0001 20 100 010
authority		0200 00 320 021	1	0100 00 210 011
away		3002 00 000 220	1	2001 00 000 110
B		0200 00 320 021	1	0100 00 210 011
bad		0002 30 100 020	1	0001 20 100 010
ball		0020 02 200 310	1	0020 02 200 320
banana		0000 10 300 000	1	0000 10 200 000

be	0000 00 020 200	1	0000 00 020 200
beast	0000 10 300 000	1	0000 00 200 000
been	0000 00 000 200	1	0000 00 000 200
bilingual	0002 30 100 020	1	0001 20 100 010
blackberry	0200 10 300 000	1	0100 10 200 000
blood	0200 10 320 021	1	0100 10 210 011
bodily	3002 30 100 121	1	3001 30 100 111
bones	0200 00 320 021	1	0100 00 210 011
bore	0000 00 200 310	1	0000 00 200 320
born	0002 30 200 220	1	0001 20 100 120
breath	0200 00 320 021	1	0100 00 210 011
breed	0020 02 200 310	1	0020 02 200 320
bridge	0000 00 200 310	1	0000 00 200 320
brightness	0200 00 320 021	1	0100 00 210 011
brilliance	0200 00 320 021	1	0100 00 210 011
burnt-offering	0000 00 300 000	1	0000 00 200 000
but	0000 00 000 000	2	0000 00 000 000
buy	0000 00 200 310	1	0000 00 200 320
by	3120 02 020 220	1	2210 01 010 110
CALL	0220 02 200 312	1	0120 02 200 320
Cambridge	0200 20 320 001	1	0100 20 210 001
can	0220 02 320 333	1	0120 02 210 232
candidates	0200 00 320 021	1	0100 00 210 011
cannot	0020 02 000 312	1	0020 02 000 222
catholic	0002 30 300 020	1	0001 20 300 010
chapter	0000 00 300 000	1	0000 00 200 000
charged	0002 30 100 230	1	0001 20 100 320
classical	0002 30 100 020	1	0001 20 100 010
classification	0200 00 320 021	1	0100 00 210 011
clear	0002 10 120 230	1	0001 20 120 322
clusters	0220 02 300 333	1	0120 02 300 333
cold	0202 30 320 021	1	0101 20 310 011
collection	0200 00 320 021	1	0100 00 210 011
comes	0020 02 000 212	1	0020 02 000 222
commenced	0022 32 100 232	1	0021 22 100 322
communion	0200 00 320 021	1	0100 00 210 011
conclusion	0200 00 320 021	1	0100 00 210 011
condition	0200 00 320 331	1	0100 00 310 331
confusions	0200 00 320 021	1	0100 00 210 011
conjectural	0002 30 100 020	1	0001 20 100 010
connect	0020 02 020 212	1	0020 02 020 322
connection	0200 00 320 021	1	0100 00 210 011
connotation	0200 00 320 021	1	0100 00 210 011
consecrated	0002 30 100 230	1	0001 20 100 320
contained	0002 30 100 230	1	0001 20 100 320
contempt	0200 00 320 021	1	0100 00 210 011
context	0200 00 320 021	1	0100 00 210 011
contrasted	0022 32 100 232	1	0021 22 100 322

contribution	0200 00 320 021	1	0100 00 210 011
convincing	0102 30 130 220	1	0201 20 120 110
could	0020 02 000 312	1	0020 02 000 222
course	0220 02 200 312	1	0120 02 200 322
Cypriot	0002 30 300 020	1	0001 20 300 010
Czech	0002 30 300 020	1	0001 20 300 010
DEATH	0200 30 320 001	1	0100 20 310 001
decipherment	0200 00 320 021	1	0100 00 210 011
declaring	0100 00 210 200	1	0200 00 220 100
deems	0000 00 000 210	1	0000 00 000 220
definition	0200 00 320 021	1	0100 00 210 011
denied	0002 30 100 210	1	0001 20 100 320
denotation	0200 00 320 021	1	0100 00 210 011
department	0200 20 320 001	1	0100 20 310 001
describes	0000 00 000 210	1	0000 00 000 220
description	0200 00 320 021	1	0100 00 210 011
destruction	0200 00 320 021	1	0100 00 210 011
detail	0200 00 320 331	1	0100 00 310 331
determine	0000 00 020 210	1	0000 00 020 320
different	0002 30 100 020	1	0001 20 100 010
differently	3000 10 000 121	1	2000 10 000 111
dim	0000 10 120 230	1	0000 20 120 322
discussed	0022 32 000 232	1	0021 22 000 322
discussion	0200 00 320 021	1	0100 00 210 011
dispelled	0002 30 200 230	1	0001 20 100 320
diverge	0020 02 020 212	1	0020 02 020 322
divinity	0200 00 320 021	1	0100 00 210 011
division	0200 00 320 021	1	0100 00 210 011
do	0020 02 020 312	1	0020 02 020 322
doctrines	0200 00 320 021	1	0100 00 210 011
doing	0100 00 230 223	1	0100 00 230 111
domestication	0200 00 320 021	1	0100 00 210 011
down	3302 00 320 223	1	3101 00 320 333
dust	0200 00 320 021	1	0100 00 210 011
dustbin	0000 00 300 000	1	0000 00 200 000
ELECTRICITY	0200 00 320 021	1	0100 00 210 011
electroscope	0000 00 300 000	1	0000 00 200 000
elements	0200 00 320 021	1	0100 00 210 011
else	0021 00 200 000	0	0012 00 100 000
emphasis	0200 00 320 021	1	0100 00 210 011
enjoy	0000 00 020 210	1	0000 00 020 320
enough	2002 20 300 021	1	2001 10 100 011
enthusiastic	0002 30 100 020	1	0001 20 100 010
entire	0000 30 300 020	1	0000 20 300 010
establish	0000 00 020 210	1	0000 00 020 320
everyday	0002 30 100 020	1	0001 20 100 010
evolutionary	0000 30 100 020	1	0000 20 100 010

exactly	3000	10	000	121	1	2000	10	000	111
examination	0200	30	320	021	1	0100	20	310	011
example	0200	00	320	021	1	0100	00	210	011
exciting	0102	30	130	220	1	0201	20	120	110
exists	0020	02	000	212	1	0020	02	000	222
explaining	0300	00	310	223	1	0300	00	230	111
exposure	0200	00	320	021	1	0100	00	310	011
extension	0200	00	320	021	1	0100	00	310	011
FACTOR	0000	00	300	000	1	0000	00	200	000
faculty	0200	30	320	021	1	0100	20	310	011
faded	0022	22	100	232	1	0021	22	100	332
failures	0200	00	320	021	1	0100	00	210	011
faithful	0002	30	100	020	1	0001	20	100	010
familiarity	0200	00	320	021	1	0100	00	210	011
fantasies	0200	00	320	021	1	0100	00	210	011
far	2002	10	100	020	1	2001	10	100	010
feel	0220	02	200	312	1	0120	02	200	322
field	0000	10	300	000	1	0000	20	300	000
finally	3000	10	000	121	1	2000	10	000	111
finding	0100	00	210	200	1	0200	00	220	100
fire	0200	10	320	331	1	0100	10	310	331
first	3002	30	300	121	1	2001	20	300	111
first-born	0002	30	300	020	1	0001	20	300	010
flames	0200	00	320	021	1	0100	00	210	011
flashing	0302	30	330	223	1	0301	20	330	111
flights	0200	00	320	021	1	0100	00	210	011
focussed	0022	32	100	232	1	0021	22	100	322
for	0120	02	020	220	1	0210	01	010	110
formal	0002	30	100	020	1	0001	20	100	010
four	0202	30	320	021	1	0101	20	310	011
from	0120	02	020	220	1	0210	01	010	110
GALAXY	0000	00	300	000	1	0000	00	200	000
gentlemen	0200	00	320	021	1	0100	00	210	011
gift	0000	00	300	000	1	0000	00	300	000
giving	0300	30	330	223	1	0300	20	330	111
go	0020	02	220	232	1	0020	02	230	332
God	0200	00	320	021	1	0100	00	210	011
godhead	0200	00	320	021	1	0100	00	210	011
grammar	0200	00	320	021	1	0100	00	310	011
grandeur	0200	00	320	021	1	0100	00	210	011
guessing	0300	00	310	223	1	0300	00	330	111
HAS	0020	02	000	312	1	0020	02	000	222
have	0020	02	020	312	1	0020	02	020	322
having	0100	00	230	200	1	0100	00	230	100
he	0000	00	100	001	1	0000	00	200	001
head	0000	00	200	310	1	0000	00	200	320

herculean	0002 30 100 020	1	0001 20 100 010
here	2202 10 200 021	1	2101 10 100 011
herself	0200 00 320 020	1	0100 00 310 010
hieroglyphic	0202 30 320 021	1	0101 20 310 011
himself	0200 00 320 020	1	0100 00 310 010
hopes	0220 02 300 333	1	0120 02 300 333
how	0001 00 000 001	1	0002 00 000 001
however	0011 00 000 000	1	0012 00 000 000
I	0000 00 100 001	1	0000 00 200 001
if	0001 00 000 000	1	0002 00 000 000
imagination	0200 00 320 021	1	0100 00 210 011
immersion	0200 00 320 021	1	0100 00 310 011
importance	0200 00 320 021	1	0100 00 210 011
important	0000 30 100 020	1	0000 20 100 010
in	3122 02 020 220	1	2211 01 010 110
incorporate	0000 00 020 210	1	0000 00 020 320
indeed	0000 22 223 021	1	0000 11 111 011
individual	0002 30 300 020	1	0001 20 300 010
inference	0200 00 320 021	1	0100 00 210 011
innumerable	0002 30 300 020	1	0001 20 100 010
intension	0200 00 320 021	1	0100 00 210 011
interconnected	0002 30 100 020	1	0001 20 100 010
interested	0002 30 100 230	1	0001 20 100 320
interests	0200 00 300 211	1	0100 00 200 333
into	0120 02 020 220	1	0210 01 010 110
investigation	0200 00 320 021	1	0100 00 310 011
involved	0002 30 100 230	1	0001 20 100 320
iridescent	0002 30 100 020	1	0001 20 100 010
is	0020 02 000 312	1	0020 02 000 222
it	0200 00 120 021	1	0100 00 210 011
its	0000 00 100 000	1	0000 00 100 000
itself	0200 00 320 020	1	0100 00 310 010
JOHN	0200 02 220 232	1	0100 02 230 332
journals	0200 00 320 021	1	0100 00 210 011
joyous	0002 30 100 020	1	0001 20 100 010
KIND	0200 30 300 020	1	0100 20 300 010
know	0020 02 020 212	1	0020 02 020 322
knowing	0300 30 330 223	1	0300 20 330 111
knowledge	0200 00 320 021	1	0100 00 210 011
LAMBDA	0200 10 320 021	1	0100 10 310 011
language	0200 00 320 021	1	0100 00 310 011
larger	0002 30 100 020	1	0001 20 100 010
learn	0020 02 020 212	1	0020 02 020 322
leaves	0220 02 320 333	1	0120 02 310 333
lengthy	0002 30 100 020	1	0001 20 100 010

herculean	0002	30	100	020	1	0001	20	100	010
here	2202	10	200	021	1	2101	10	100	011
herself	0200	00	320	020	1	0100	00	310	010
hieroglyphic	0202	30	320	021	1	0101	20	310	011
himself	0200	00	320	020	1	0100	00	310	010
hopes	0220	02	300	333	1	0120	02	300	333
how	0001	00	000	001	1	0002	00	000	001
however	0011	00	000	000	1	0012	00	000	000
I	0000	00	100	001	1	0000	00	200	001
if	0001	00	000	000	1	0002	00	000	000
imagination	0200	00	320	021	1	0100	00	210	011
immersion	0200	00	320	021	1	0100	00	310	011
importance	0200	00	320	021	1	0100	00	210	011
important	0000	30	100	020	1	0000	20	100	010
in	3122	02	020	220	1	2211	01	010	110
incorporate	0000	00	020	210	1	0000	00	020	320
indeed	0000	22	223	021	1	0000	11	111	011
individual	0002	30	300	020	1	0001	20	300	010
inference	0200	00	320	021	1	0100	00	210	011
innumerable	0002	30	300	020	1	0001	20	100	010
intension	0200	00	320	021	1	0100	00	210	011
interconnected	0002	30	100	020	1	0001	20	100	010
interested	0002	30	100	230	1	0001	20	100	320
interests	0200	00	300	211	1	0100	00	200	333
into	0120	02	020	220	1	0210	01	010	110
investigation	0200	00	320	021	1	0100	00	310	011
involved	0002	30	100	230	1	0001	20	100	320
iridescent	0002	30	100	020	1	0001	20	100	010
is	0020	02	000	312	1	0020	02	000	222
it	0200	00	120	021	1	0100	00	210	011
its	0000	00	100	000	1	0000	00	100	000
itself	0200	00	320	020	1	0100	00	310	010
JOHN	0200	02	220	232	1	0100	02	230	332
journals	0200	00	320	021	1	0100	00	210	011
joyous	0002	30	100	020	1	0001	20	100	010
KIND	0200	30	300	020	1	0100	20	300	010
know	0020	02	020	212	1	0020	02	020	322
knowing	0300	30	330	223	1	0300	20	330	111
knowledge	0200	00	320	021	1	0100	00	210	011
LAMBDA	0200	10	320	021	1	0100	10	310	011
language	0200	00	320	021	1	0100	00	310	011
larger	0002	30	100	020	1	0001	20	100	010
learn	0020	02	020	212	1	0020	02	020	322
leaves	0220	02	320	333	1	0120	02	310	333
lengthy	0002	30	100	020	1	0001	20	100	010

less	3202 30 320 131	1	3101 30 310 111
libraries	0200 00 320 021	1	0100 00 210 011
light	0202 10 320 231	1	0101 20 310 331
likely	0000 30 100 020	1	0000 20 100 010
linear	0002 30 100 020	1	0001 20 100 010
literally	3000 10 000 121	1	2000 10 000 111
little	3200 30 320 131	1	3100 30 310 111
logical	0002 30 100 020	1	0001 20 100 010
logician	0000 00 300 000	1	0000 00 200 000
luckless	0002 30 100 020	1	0001 20 100 010
MADE	0002 30 100 230	1	0001 20 100 320
magic	0202 30 320 021	1	0101 20 310 111
make	0000 00 020 210	1	0000 00 020 320
malaria	0200 00 320 021	1	0100 00 210 011
market-place	0000 00 300 000	1	0000 00 200 000
may	0220 02 320 133	1	0120 02 210 233
me	0200 00 120 020	1	0100 00 210 010
mean	0200 10 300 230	1	0100 20 300 220
meaning	0300 30 330 223	1	0300 20 330 111
means	0200 00 300 211	1	0100 00 200 333
melange	0000 00 300 000	1	0000 00 200 000
men	0200 00 320 021	1	0100 00 210 011
mental	0002 30 100 020	1	0001 20 100 010
metal	0202 30 320 021	1	0101 20 310 011
metaphor	0200 00 320 021	1	0100 00 210 011
method	0200 00 320 021	1	0100 00 210 011
middle	0000 30 300 020	1	0000 20 300 010
metaphysical	0002 30 100 020	1	0001 20 100 010
modern	0002 30 100 020	1	0001 20 100 010
moment	0000 00 300 000	1	0000 00 200 000
more	3202 30 320 131	1	3101 30 310 111
moreover	0000 22 223 021	1	0000 11 111 011
morning	0200 10 320 021	1	0100 10 310 011
mote	0200 00 320 021	1	0100 00 210 011
movement	0200 00 320 021	1	0100 00 210 011
moving	0302 30 330 223	1	0301 20 330 111
must	0220 02 320 133	1	0120 02 210 233
muster	0200 00 200 312	1	0120 02 200 322
my	0000 00 100 000	1	0000 00 100 000
myth	0200 00 320 021	1	0100 00 210 011
NAMES	0200 00 320 021	1	0100 00 210 011
naturally	3000 10 000 121	1	2000 10 000 111
net	0202 10 300 230	1	0201 20 300 320
new	0002 30 100 020	1	0001 20 100 010
next	0000 30 300 020	1	0000 20 300 010
no	1000 10 100 000	1	1000 10 100 000
nodded	0020 02 000 212	1	0020 02 000 322

noncommittal	0002 30 100 020	1	0001 20 100 010
not	1102 10 110 120	1	1101 10 110 110
notably	3000 10 000 121	1	2000 10 000 111
now	3002 10 000 121	1	2001 10 000 111
OBSERVABLE	0002 30 300 020	1	0001 20 300 010
occurred	0020 02 000 212	1	0020 02 000 322
of	0120 02 020 220	1	0210 01 010 110
on	3120 02 020 220	1	2210 01 010 110
once	2202 10 000 121	1	2101 10 000 111
one	0202 20 320 021	1	0101 20 310 011
only	1001 10 100 101	1	1002 10 100 101
operations	0200 00 320 021	1	0100 00 210 011
or	0000 00 000 000	2	0000 00 000 000
order	0200 00 320 331	1	0100 00 310 331
originally	3000 10 000 121	1	2000 10 000 111
others	0200 00 320 021	1	0100 00 210 011
out	3002 00 000 220	1	2001 00 000 110
outburst	0000 00 300 000	1	0000 00 200 000
outside	0202 00 320 021	1	0101 00 310 011
over	3120 02 020 220	1	2210 01 010 110
own	0200 10 300 230	1	0100 20 300 220
P	0200 00 320 021	1	0100 00 210 011
paper	0200 10 320 021	1	0100 10 310 011
paragraph	0000 00 300 000	1	0000 00 200 000
pardon	0200 00 320 331	1	0100 00 310 331
part	0220 12 320 333	1	0120 12 310 333
pass	0220 02 320 333	1	0120 02 210 333
peak	0200 00 300 020	1	0100 00 300 010
pencil	0000 00 300 000	1	0000 00 200 000
permanent	0002 30 100 020	1	0001 20 100 010
perverse	0002 30 100 020	1	0001 20 100 010
place	0200 00 320 331	1	0100 00 310 331
placed	0002 30 100 210	1	0001 20 100 320
plant	0200 00 320 331	1	0100 00 310 331
play	0220 02 320 333	1	0120 02 310 333
point	0200 00 320 331	1	0100 00 310 331
points	0200 00 300 211	1	0100 00 200 333
polemical	0002 30 100 020	1	0001 20 100 010
possible	0002 30 100 020	1	0001 20 100 010
practicable	0002 30 100 020	1	0001 20 100 010
present	0202 10 300 230	1	0101 20 300 320
presented	0002 30 100 210	1	0001 20 100 320
press	0220 02 320 333	1	0120 02 310 333
primitive	0002 30 300 020	1	0001 20 300 010
principle	0200 00 320 021	1	0100 00 210 011
private	0002 30 300 020	1	0001 20 300 010
process	0000 00 200 310	1	0000 00 200 320

proconsul	0000 00 300 000	1	0000 00 200 000
proper	0002 30 300 020	1	0001 20 100 010
psychology	0200 00 320 021	1	0100 00 210 011
public	0002 30 300 020	1	0001 20 100 010
published	0022 32 100 232	1	0021 22 100 322
purposes	0200 00 300 211	1	0100 00 200 333
pure	0002 30 100 020	1	0001 20 100 010
pushes	0220 02 300 333	1	0120 02 300 333
put	0000 10 020 210	1	0000 20 020 320
Pylos	0200 00 320 021	1	0100 00 210 011
QUITE	1000 10 000 100	1	2000 10 000 100
RADICAL	0002 30 300 020	1	0001 20 300 010
reach	0000 00 200 310	1	0000 00 200 320
read	0002 30 020 210	1	0001 20 120 320
readers	0200 00 320 021	1	0100 00 210 011
recourse	0200 00 220 021	0	0100 00 210 011
references	0200 00 320 021	1	0100 00 210 011
reflect	0020 02 020 212	1	0020 02 020 322
religion	0200 00 320 021	1	0100 00 210 011
remind	0000 00 020 210	1	0000 00 020 320
reminder	0000 00 300 000	1	0000 00 200 000
remove	0220 02 300 333	1	0120 02 300 333
reprint	0200 00 300 211	1	0100 00 200 333
result	0220 02 300 333	1	0120 02 300 333
right	0202 10 320 231	1	0101 20 310 321
rim	0000 00 300 000	1	0000 00 200 000
risked	0002 00 000 210	1	0001 00 000 320
rites	0200 00 320 021	1	0100 00 210 011
river	0000 00 300 000	1	0000 00 200 000
root	0020 02 300 310	1	0010 02 300 320
rule	0220 02 200 312	1	0120 02 200 322
SACRIFICE	0200 00 320 331	1	0100 00 210 331
same	0000 00 200 000	1	0000 00 300 000
science	0200 00 320 021	1	0100 00 210 011
scientific	0002 30 100 020	1	0001 20 100 010
script	0200 00 300 000	1	0100 00 300 000
second	0202 10 300 320	1	0101 20 300 320
see	0020 02 200 310	1	0020 02 200 320
seems	0000 00 000 210	1	0000 00 000 210
seen	0002 30 300 220	1	0001 20 100 120
self	0200 00 220 020	1	0100 00 210 010
sensation	0200 00 320 021	1	0100 00 210 011
sense	0200 00 320 331	1	0100 00 210 331
sentence	0000 00 320 331	1	0000 00 210 331
several	0002 00 100 021	1	0001 00 300 011
shine	0220 02 320 332	1	0120 02 310 332

short	0202	10	320	232	1	0101	20	310	321
should	0020	02	000	112	1	0020	02	000	222
silver	0202	00	320	331	1	0101	00	210	331
situation	0200	00	320	021	1	0100	00	210	011
six	0202	30	320	021	1	0101	20	310	011
slain	0002	30	100	220	1	0001	20	100	120
sold	0002	30	200	230	1	0001	20	100	320
solid	0002	30	300	020	1	0001	20	300	010
solved	0002	30	200	230	1	0001	20	100	320
something	0200	00	320	021	1	0100	00	210	011
sort	0000	00	200	310	1	0000	00	200	320
space	0200	00	320	331	1	0100	00	210	331
speak	0020	02	020	212	1	0020	02	020	322
species	0200	00	320	021	1	0100	00	210	011
spectacle	0000	00	300	000	1	0000	00	200	000
spirits	0200	00	300	211	1	0100	00	200	333
spun	0022	32	100	232	1	0021	22	100	322
stand	0020	02	200	310	1	0020	02	200	320
standpoint	0000	00	300	000	1	0000	00	200	000
star	0020	02	200	310	1	0020	02	200	320
stars	0220	02	300	333	1	0120	02	300	333
statement	0000	00	300	000	1	0000	00	200	000
still	0202	30	300	231	1	0101	30	300	331
stop	0020	02	200	310	1	0020	02	200	320
subject	0200	10	200	230	1	0100	20	200	320
subsequent	0000	30	100	020	1	0000	20	100	010
such	0202	00	100	020	1	0101	00	100	010
sun	0000	00	200	000	1	0000	00	200	000
suppose	0020	02	020	212	1	0020	02	020	322
symposiast	0000	00	300	200	1	0000	00	200	000
system	0200	00	320	021	1	0100	00	210	011
TAKE	0020	02	020	212	1	0020	02	020	322
takes	0020	02	000	212	1	0020	02	000	222
talk	0220	02	320	333	1	0120	02	310	333
Tarpeia	0200	00	320	021	1	0100	00	210	011
task	0200	00	300	000	1	0100	00	300	000
tearing	0100	30	130	220	1	0200	20	120	110
term	0200	00	200	310	1	0100	00	200	320
terminological	0002	30	100	020	1	0001	20	100	010
terminology	0200	00	320	021	1	0100	00	210	011
text	0200	00	320	021	1	0100	00	210	011
that	0203	21	121	021	1	0103	21	312	011
the	0000	00	100	000	1	0000	00	100	000
their	0000	00	100	000	1	0000	00	100	000
them	0200	00	120	020	1	0100	00	210	010
thereafter	2000	30	000	123	1	2000	10	000	111
these	0200	20	120	021	1	0100	20	310	011
things	0200	00	320	021	1	0100	00	210	011

this	0200 20 120 021	1	0100 20 310 011
thought	0220 32 320 233	1	0120 22 310 333
through	3122 02 020 220	1	2211 01 010 110
throwing	0302 00 310 223	1	0301 00 230 111
tie	0020 02 200 310	1	0020 02 200 320
tiny	0002 30 100 020	1	0001 20 100 010
to	2100 00 010 220	1	2200 00 010 110
too	1000 10 000 002	1	2000 10 000 001
tooth	0000 00 300 000	1	0000 00 200 000
topics	0200 00 320 021	1	0100 00 210 011
torn	0002 30 000 220	1	0001 20 000 120
toyed	0020 02 000 212	1	0020 02 000 322
traditional	0002 30 100 020	1	0001 20 100 010
translations	0200 00 320 021	1	0100 00 210 011
tried	0022 32 100 232	1	0021 22 100 322
truths	0200 00 320 021	1	0100 00 210 011
two	0202 30 320 021	1	0101 20 310 011
UNDERSTAND	0020 02 020 212	1	0020 02 020 322
unfortunately	3000 10 000 121	1	2000 10 000 111
unlikely	0002 30 100 020	1	0001 20 100 010
up	3120 02 020 220	1	2210 01 010 110
upon	0120 02 020 220	1	0210 01 010 110
us	0200 00 120 020	1	0100 00 210 010
use	0200 00 320 331	1	0100 00 210 331
VARIES	0000 00 000 210	1	0000 00 000 220
various	0002 30 100 020	1	0001 20 100 010
Veda	0000 00 300 000	1	0000 00 200 000
venture	0020 02 200 310	1	0020 02 200 320
verigiability	0200 00 320 021	1	0100 00 210 011
verification	0200 00 320 021	1	0200 00 310 011
very	1000 10 200 000	1	3000 10 100 000
view	0200 00 200 310	1	0100 00 300 320
virgin	0002 30 300 020	1	0001 30 300 010
visible	0002 30 100 020	1	0001 20 100 010
vision	0200 00 320 021	1	0100 00 210 011
vote	0020 02 200 310	1	0020 02 200 320
WAS	0020 02 000 312	1	0020 02 000 222
watch	0220 02 320 333	1	0120 02 310 333
way	0000 00 320 021	1	0000 00 210 011
ways	0200 00 320 021	1	0100 00 210 011
we	0000 00 100 001	1	0000 00 200 001
went	0020 02 020 212	1	0020 02 020 322
what	0210 01 001 011	1	0110 01 002 011
wheeled	0002 30 100 230	1	0001 20 100 320
when	0001 01 011 001	1	0002 01 012 001
where	0001 01 011 001	1	0002 01 012 001

whether	0000 00 001 000	1	0000 00 002 000
which	0200 21 121 011	1	0100 21 112 011
while	0001 01 311 001	1	0002 01 212 001
white	0202 30 320 021	1	0101 20 310 011
who	0000 01 001 001	1	0000 01 002 001
whom	0200 01 021 010	1	0100 01 012 010
why	0000 01 001 001	1	0000 01 002 001
will	0220 02 220 333	1	0120 02 210 232
winds	0200 00 300 211	1	0100 00 200 331
with	0122 02 020 220	1	0211 01 010 110
without	0122 02 020 222	1	0211 01 010 111
works	0220 02 320 333	1	0120 02 310 333
would	0020 02 000 113	1	0020 02 000 222
would-be	0000 30 100 000	1	0000 20 100 000
written	0000 30 100 220	1	0000 20 100 120
wrong	0202 10 300 230	1	0101 20 300 320
wry	0000 30 100 000	1	0000 20 100 000
YEAR	0000 00 300 000	1	0000 00 200 000
yet	2000 00 000 121	1	2000 00 000 111
you	0200 00 120 021	1	0100 00 210 011
your	0000 00 100 000	1	0000 00 100 000
'RE	0000 00 000 100	0	0000 00 000 200
's	0200 00 000 100	0	0200 00 000 200
1950	0200 00 320 021	1	0100 00 210 011
(	0110 01 000 000	0	0210 01 000 000
)	0220 02 000 000	0	0330 03 000 000
,	2222 22 222 222	2	3333 33 333 333
;	0000 00 000 002	0	0000 00 000 003
:	0000 00 000 001	0	0000 00 000 003
!	0000 00 000 002	0	0000 00 000 003
?	0000 00 000 002	0	0000 00 000 003
.	0000 00 000 002	0	0000 00 000 003

Participation Classes of Substituent Types

Adverbial Group	0000 30 000 121	1	0000 10 000 111
Adverbial O-Clause			
Normal Case	3009 30 200 021	1	2101 10 100 011
Relative as Dependent	0000 01 220 000	1	0000 01 110 000
Adverbial S-Clause	0200 00 020 021	1	0100 00 010 033
Adverbial Subjunct	0000 20 200 021	1	0000 10 100 011
Adjunctive Group	0002 00 300 021	1	0001 00 100 011
Adjunctive S-Clause	0000 00 200 020	1	0000 00 100 010
Nominal Group	0222 02 020 021	1	0111 01 010 011
Nominal O-Clause			
Normal Case	0202 20 200 021	1	0101 10 110 011
Relative as Dependent	0000 21 000 000	1	0000 11 000 000
Nominal Subjunct	0000 00 020 021	1	0000 00 010 011
Operative Group	0020 02 000 013	1	0020 01 000 022
Operative S-Clause	0020 02 020 012	1	0020 02 020 122
Full Clause	0002 00 002 003	1	0001 00 001 001

** a "relative dependent" is an Adjunctive S-Clause, or any substituent itself having a relative dependent.

APPENDIX II.

M.L. 162

A SYSTEM FOR PRODUCING ENGLISH SYNTACTIC DICTIONARY ENTRIES

Y. Wilks

Eloise Rigby

 September 1962

Cambridge Language Research Unit,
20, Millington Road,
Cambridge, England.

A System for producing English Syntactic Dictionary Entries

The method of 'sorting frames' set out here has a limited purpose: to extend the Parker-Rhodes Syntax Dictionary (PRSD) without having to decide the Governor-Dependent information entry for each word digit by digit. We have worked within the general framework of the PRSD method to the extent that we have considered the 12 substituent types (for English words) to be adequately determined by the list of examples supplied by its author. † It should be noticed that defects in the original method of classifying some kinds of substituents persist, e.g. all verb groups without noun objects are considered to bracket in type (10)‡, hence we have no distinction between the functions of adjectives and past participles in this type.

Basing this work upon the examples has to a large extent made it independent of the PRSD itself. Thus our governor-dependent entries for words occurring in the PRSD may be different from those to be found there. The present work thus checks

---

† For the list, see Appendix I. For a discussion of the substituent types themselves see Parker-Rhodes' et al, "A Lattice Model of Syntactic Description", M.L. 147, "Supplement I. to M.L. 147", M.L. 154 and "How to Construct a Syntactic Dictionary Entry", M.L. 153.

‡ The twelve substituent types will be referred to in this way throughout.

existing entries as well as producing new entries on a large scale. Experiments will show whether the entries produced by the present method bracket better than the previous entries; the method, however, as opposed to the original entry by entry approach, has very considerable advantages: once words have been assigned to parts of speech, consistent detailed entries can be allocated in a quasi-automatic way.

General Method

The general method is to sort alphabetically punched word cards into heaps, each consisting of words with a common governor-dependent entry; this entry can then be gang-punched through the pack from a master card.

The sorting is done by asking a series of questions, A, which divide the words into five classes corresponding crudely to parts of speech. Further series of questions are then asked for each of these classes, and the answers to these lead to tables which give the required governor-dependent dictionary entries. One question in A rejects words, classified as 'fancy', which will require individual entries. It is thought that there will not be many of these.

Questions A

Question	If the answer is "yes"
1. Is the word 'fancy', i.e. i) subjunction ii) conjunction iii) part of the word "to be" iv) pronoun v) auxiliary verb?	R (reject)
2. Is it a preposition?	See Table III.
3. Can it have any adverb uses?	" " IV.
4. " " " " verb "	" " II.
5. " " " " noun "	" " I.
6. " " " " adjective uses?	" " V.
Any remaining word (e.g., interjection)	R

Table I

Questions	Answer: Yes	Answer: No
1. Has the word noun uses?	N	N
2. Has it adjective uses?	A	A
3. If a verb, can it stand alone with subject but without qualifier or object?	I	I
4. Must it stand (e.g., "exists") without an object.	I + (R)	
5. If an adjective, is it only one if preceded by an adverb? (e.g., "bred")	A + (R)	
6. If a noun, can it be used without an article or other modifier?	G	G
7. If G, can it be used without an article or other modifier after a preposition in type (2a)?	G	G –

Note on the coding of adjectives

A word is an adjective if it can qualify any noun at all, even as an article. If it can only function thus it is called a 'nounish' (n) adjective, written with subscript, A_n. If it can also plausibly be used after "when" or "if" (e.g. "when drunk I drove badly") and can be qualified by an adverb (e.g. "very drunk") it is called a 'plain' (p) adjective, written with superscript, $\overset{p}{A}$, In the tables alternative readings of governor-dependent entries for those two are given by two figures, one above the other: for A_n read the lower, for $\overset{p}{A}$ the upper. If the adjective is neither 'nounish' or 'plain' the word is rejected as 'fancy'.

A_n adjectives can in general be used to qualify gerunds ("man $\overset{p}{A}$ eating tiger") while cannot. Note also: for gerunds the criteria for adjectives are different:

If A_n they can qualify any noun

If $\overset{p}{A}$ they can qualify any noun and be qualified by an adverb.

Examples

a. "Cypriot". Ref. table I.
Coding: $\overset{p}{NAG}$ (I/I irrelevant)

b. "occurs". Ref. table II. (nullified by reject below)
Coding: NAI+, I+ = R.
(G/G irrelevant)

c. "others". Ref. table I.

Coding: NAG (J/I irrelevant)

Table I

Code	Dictionary Entry
NAG	0 1 0 $\begin{matrix}1\\0\end{matrix}$ $\begin{matrix}2\\0\end{matrix}$ 0 3 1 0 $\begin{matrix}1\\0\end{matrix}$ 1 1
NAG	0 1 0 0 0 0 2 1 0 0 1 1
NAG	$\begin{matrix}(0)\\1\\(1)\end{matrix}$ 0 $\begin{matrix}1\\0\end{matrix}$ $\begin{matrix}2\\0\end{matrix}$ 0 3 0 0 $\begin{matrix}1\\0\end{matrix}$ $\begin{matrix}1\\0\end{matrix}$ 0
NAG	$\begin{matrix}(0)\\1\\(0)\end{matrix}$ 0 0 0 0 2 0 0 0 0 0

Note on the Code of G-

Alternative readings for G- are given in brackets; there will always be two such entries, to account for both 'n' and 'p' adjectives, even where the two entries are the same. Thus the word "hospital", in Table I, will have the coding NAG- and the entry 0100 0030 0000, that is, the lower alternative for the coding NAG.

Table II

Questions		Answer
1 - 7.	ASK THE QUESTIONS GIVEN UNDER TABLE I	
8.	Is the verb a present tense first person singular form?	VI
9.	Is it a present tense third person singular?	V2
10.	Is it a past tense third person singular?	V3
11.	Is it a past tense and past participle? *	V3'
12.	Is it a past participle only?	V4
13.	Is it a gerund?	V5

* i.e. of the same verb.

Table II

Code	V1	V2	V3
NAIG	$010^{12}_{00}2310333$	$012^{12}_{00}2310^{3}_{2}33$	
NAIG	$\begin{matrix}(0)\\(1)\end{matrix}010^{12}_{00}230C3^{3}_{2}2$		$\begin{matrix}(0)\\(1)\end{matrix}012^{12}_{00}2300^{33}_{22}2$
NAIG	$010^{12}_{00}0310331$		

Table II - continued

NAIG	$\begin{matrix}(0)\\ 010^{12}_{00}03003^{3}_{2}0\\ (1)\end{matrix}$		
NAIG	010002210333	012002210233	
NAIG	(0) 010002200322 (0)		
NAIG	010000210331		
NAIG	(0) 010000200320 (0)		
NAI	$000^{12}_{00}21003^{3}_{2}2$		
NAI	$000^{12}_{00}01003^{3}_{2}0$		$000^{12}_{00}0100^{33}_{22}0$
NAI	000002000322		002002000222
NAI	000000000320		000000000220

Table II - continued

Code	V3'	V4	V5
NAIG		does not occur	does not occur
NAIG		" " "	" " "
NAIG			$0301^{2*}_{0}0330111$

Table II - continued

NAIG			does not occur
NAIG		does not occur	" " "
NAIG		" " "	" " "
NAIG			030000230111
NAIG			does not occur
NAI	$002^{12}_{00}21003^{3}_{2}2$	does not occur	does not occur
NAI	$000^{12}_{00}01003^{3}_{2}0$		$0201^{2*}_{0}0120110$
NAI	002002000322	does not occur	does not occur
NAI	000000000320	000000000100	020000020110

Note

Only cases which have actually occurred in the dictionary sample to date have been considered. Blanks will be filled in when necessary. Cases which cannot occur are indicated.

↓↑ N.B. For gerunds which are adjectives see the Note on the coding of adjectives as 'p' and 'n' above.

Assumption: for all gerunds.

If they are nouns they are coded G, if verbs are coded I; they all depend in type (4), do not depend in type (2b), and depend in type (11) (save I+ gerunds like "occurring").

Table III (Prepositions)

BASIC ENTRY:	2	2	0	0	0	0	0	1	0	1	1	0
(adjustments)		N	V	A	A	V	N			V	V	N
							A					V

Assumptions:

All prepositions are adverbs.

No prepositions are gerunds.

Table IV (Adverbs)

Additional question

Can it modify other adverbs? Yes = 3 in (1).

BASIC ENTRY:	2	0	0	0	1	0	0	0	0	1	1	0
(adjustments)		N	V	A	A	V	N	N		V	V	N
							A					V

Assumption:

All adverbs can qualify adjectives.

Table V (Adjectives without noun or verb uses)

They are all 'plain' adjectives.

BASIC ENTRY:	0	0	0	1	2	0	1	0	0	1	1	0

Note on Tables III, IV and V

For these no coding is used. For Tables III and IV basic entries are given which are modified if the word has in addition noun, verb or adjective functions. The letters N,V,A are marked under the relevant columns for adjustment in such cases according to Tables I, II, and V.

EXAMPLES

1) "arm" Table II = 0 1 0 0 0 2 3 0 0 3 2 2
code VI/ $N_{n}^{A}IG$

PRSD entry is 0 0 2 0 0 2 2 0 0 2 2 0

which is wrong in each of the four positions in which it differs from the table.

2) "agriculture" Table I = 0 1 0 0 0 0 2 1 0 0 1 1
code NAG

PRSD entry is 0 1 0 0 0 0 2 1 0 0 1 1

3) "of" Table III = 2 2 0 0 0 0 0 1 0 1 1 0
(simple, no N,V,A functions)

PRSD entry is 0 2 1 0 0 1 0 1 0 1 1 0

which is wrong in each of the four positions in which it differs from the table.

APPENDIX I

ENGLISH EXAMPLES FOR THE SYNTAX DICTIONARY

The interlingual name is given first, the English name second.

The governor is underlined.

0. Conjunct Group = CONJUNCT GROUP

before tea and afterwards
excellent and of good report
men, women and children
soak or rinse the clothes
I came but found no-one

1. Adverbial Group = ADVERBIAL GROUP

rather | nicely
exactly | in the middle
not | at all
excessively | in my opinion

2. Adverbial O-Clause = PREPOSITIONAL PHRASE OR PARTICIPLE CLAUSE

A) in | debt
from | whom (with relative pronoun as dependent)
among | the savage tribes
out of | jail

B) thinking | happy thoughts
man | eating | tigers
accompanying | her husband

(all these participial clauses can also be used as infinitive (gerund) clauses)

3. Adverbial S-Clause = ABSOLUTE CLAUSE

whenever | I go [he's out]
[I don't care] | what you do
however | I try [I fail]
[I shan't do it] whoever | says so

(all these have also noun function) ("what" is exceptional, as it always makes a noun clause; but it is still counted in this type)

4. Adverbial Subjunct = ADVERBIAL CLAUSE

if | pigs had wings |	
[Don't drive] when | drunk	(adjective as dependent)
when | my ship comes home	
In order that | they should try harder	

5. Adjunctive Group = ADJECTIVAL GROUP

very | pretty	
larger | than life	
punctual | to the minute	
heavy and gross | beyond belief	(two dependents here)
almost | black | on the sides	

6. Adjunctive S-Clause = RELATIVE CLAUSE

[people] who | stare and stare	
[ships] that | pass in the night	(relative pronoun zero)
[crooks] .. | I have known	

7. Nominal Group = NOUN GROUP

rather horrid old | men	(two dependents)
long | words | which no one can follow	(two dependents)
little | pitchers	
the | moon	

8. Nominal O-Clause = INFINITIVE (OR GERUND) CLAUSE

to make | dictionaries [is exhausting]	
thinking | happy thoughts [is fun]	
never to have made | mistakes [is not | to have learnt]	
to suddenly stop | talking [is rude]	
[I did that] to make | you laugh	(example of adverbial use)
[something] to make | you laugh	(example of adjectival use)
[a bun] to eat | which [is risky]	(with relative pronoun as dependent)

9. Nominal Subjunct = NOMINAL CLAUSE

[I found] that | they were equal
[She asked] when | they would arrive
[I wonder] whether that is true
[I think] .. | you're a fool (governor zero)

10. Operative Group = VERB GROUP

[I] have | come (main verb as dependent)
[The house] will have been being | built | two weeks] (adverb as dependent)
[They] soon | did
[He] perhaps | might have | already arrived (three dependents)
[That] may | never happen (two dependents)
[She] (too often) | forgets (complete verb as governor)
[That] is | absurd (to be with adjective)
[You] really | look | ridiculous (complete verb with adverb and adjective as two dependents)

11. Operative S-Clause = PREDICATE

[They] like | it
[She] loves | him
[I] often tried to cross | the little river
[He] gave | me some books (two dependents)
[You] should paint | the door white (two dependents, one an adjective)
That needs doing

12. Full Clause = FULL CLAUSE

It | is raining
It is difficult | to do that (note "it" of concord before infinitive clause)

People | like children
| Go to Hell (omission of dependent makes imperative)

[Who's there?] Me |. (omission of governor)
The more | the merrier (anomalous type : the second half may be regarded as a predicate with zero verb)

CAMBRIDGE LANG. AF61(052)-542.
RES. UNIT. EO. OAR. TR.
Cambridge. Eng.

M.L. 165
30 Mar 1963.

INTER-LINGUAL SYNTAX BRACKETTING PROGRAMS.

D. S. LINNEY.

ABSTRACT: Investigation of an inter-lingual syntax brackotting program using the Cambridge Language Research Unit's syntax dictionary for English.

CAMBRIDGE LANG. AF61(052)-542
RES. UNIT. EO. OAR. TR
Cambridge. Eng.

M.L. 165
30 Mar 1963.

INTER-LINGUAL SYNTAX BRACKETTING PROGRAMS.

D. S. LINNEY.

ABSTRACT: Investigation of an inter-lingual syntax brackotting program using the Cambridge Language Research Unit's syntax dictionary for English.

CAMBRIDGE LANG. AF61(052)-542.
RES. UNIT. EO. OAR. TR.
Cambridge. Eng.

M.L. 165
30 Mar 1963.

INTER-LINGUAL SYNTAX BRACKETTING PROGRAMS.

D. S. LINNEY.

ABSTRACT: Investigation of an inter-lingual syntax brackotting program using the Cambridge Language Research Unit's syntax dictionary for English.

CAMBRIDGE LANG. AF61(052)-542.
RES. UNIT. EO. OAR. TR.
Cambridge. Eng.

M.L. 165
30 Mar 1963.

INTER-LINGUAL SYNTAX BRACKETTING PROGRAMS.

D. S. LINNEY.

ABSTRACT: Investigation of an inter-lingual syntax brackotting program using the Cambridge Language Research Unit's syntax dictionary for English.